Towards the Development of a Web-based Alignment Platform

Examination Number: 7254047
MSc Speech and Language Processing
The University of Edinburgh

2010

Abstract

In this work, a platform is developed that makes existing sentence and word alignment tools available as web services. The tools implemented are Hunalign and GIZA++; after creating wrappers and format converters, they are embedded into a pipeline that produces a collection of word-aligned sentences from a parallel corpus provided by the user. The single components of the platform are independent of each other and therefore can be used in any order and by any web service client. The platform components are implemented in a generalisable way such that additional modules from any other sub-fields of natural language processing or completely different fields as well can be developed using methods shown in this work.

After giving a background view on the state of the art alignment techniques, the platform development itself is demonstrated and evaluated. Examples of how to use it with different clients are presented. The alignment platform is implemented as a Java servlet class that is by design usable on any operating system. It is generated using the Soaplab software suite. Its tool *acd2xml* automatically creates web service descriptions from a definition written in the ACD format that has been designed by the Emboss project (*European Molecular Biology Open Software Suite*).

Acknowledgements

I would like to thank ELDA for letting me realise this dissertation as part of my work and the University of Edinburgh for supporting me. My supervisors Victoria Arranz at ELDA and Simon King at the University of Edinburgh as well as Olivier Hamon have been a great help for me with their advice.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This work is about the efficient application of natural language processing tools. It demonstrates the development of a platform for the alignment of parallel corpora in which existing technologies are applied and combined in such a way that the production of aligned corpora can be widely automatised without demanding much technical knowledge from the user. Therefore, the primary target group comprises all users who apply tools for sentence and word alignment on a regular basis. On a wider scale, the integration of additional components makes the platform helpful for users of other tools too.

The platform is designed in a generalisable and portable manner. All its components can be combined in any order or replaced with different tools from internal or external sources. This is realised by a concept based on web services: state of the art sentence and word alignment tools are implemented as web services which, from a user perspective, means that installation and maintenance issues are no longer a hurdle. Format converters are added to ensure that data produced by one component can be used by the next component in a pipeline that leads from a plain input corpus to the desired result.

Regarding the generalisable design of the platform, its actual tasks – sentence and word alignment – serve two purposes. Apart from being an essential part of applications like machine translation systems by themselves, the implemented web services can be seen as proof of concept. Any other kind of tool can be integrated into the established platform as well and therefore it can form the base of an easily usable and efficient way to produce language resources of all kinds.

1.1 Motivation

Researchers and developers of natural language processing tools generally focus on the quality of the results. Taking the example of a sentence aligner, the tool's quality is determined by its ability to detect corresponding sentences with high accuracy. This focus is an obvious choice because an improvement in alignment quality, for instance, is expected to yield improvements in the quality of a machine translation system that applies that alignment algorithm.

However, implementing an algorithm and making it usable for an application requires addi-

tional resources as it can be a very time-consuming task on its own. In the meantime, language processing tools often are only single parts of a pipeline that lead, in this work, from a plain multi-lingual corpus via a sentence aligner and a word aligner to a set of word-aligned sentences. Even this application with relatively few steps requires two tools the user needs to install and to configure. In addition, he has to pass their input and outputs that do not share compatible formats from one component to the other. Integrating all the tools of a large processing pipeline, in the worst case, is not feasible with the resources available.

Web services are a comfortable way for users of all levels of technical expertise to access software tools in general. While the user inevitably needs to grasp the meaning of e.g. the input for an aligner, it should not be necessary that a linguistic researcher has to struggle with the installation and the syntax of a software tool. Also, tools can be restricted to certain operating systems and versions, require special libraries and other additional software that may not be available. These factors can prevent researchers from applying a tool even if it would be the first choice from a quality point of view.

From another point of view, the developers might not want or be allowed to open their tools to the community due to intellectual property rights (IPR). Web services are a way to provide software functionality while neither releasing code or data nor having to deal with potential IPR infringement. This also makes it easier for potential users to license exactly the functionality they require. A web service provided by the developer will allow the user to go just as far as the application was intended to go during development.

1.2 Objectives

In chapter 2, I will give an overview over existing alignment methods on both sentence and sub-sentence level. Chapter 3 will explains how web services work in general, which languages they use and which technologies can be used to provide them. In chapter 4, the platform development will be documented. Ideally, it will fulfil the following criteria:

- Modularity: instead of offering a statically predefined pipeline, the platform should allow users to customise it according to their needs i.e. to select single components, integrate components from other sources, and define the order of the components.
- Usability: the platform and its components should be usable with as little technical background as possible.
- Robustness:
  - Large Data: linguistic resources like corpora often are very large, e.g. the 45 million English words provided by the multi-lingual Europarl corpus version 5 (Koehn, 2005) take 278 megabytes of diskspace; the platform needs to be able to deal with these amounts of data.
  - Heavy Load: the web server should be usable by a large number of users at the same time.

- Generalisability: the platform design and component integration should be applicable for other purposes, at least in the field of natural language processing.
- Platform Independence: the platform should run on commonly used operating systems and web servers and require no additional external software packages.
- Integration: the different components should share input and output formats in order to process the outcome of each other and services provided by third parties.

1.3 Context

1.3.1 PANACEA

This work takes place in the context of PANACEA[1] (Platform for Automatic, Normalized Annotation and Cost-Effective Acquisition of Language Resources for Human Language Technologies), a joint project by several European universities and companies: the *Universitat Pompeu Fabra* (Barcelona, Spain), the *Consiglio Nazionale delle Ricerche – Istituto di Linguistica Computazionale* (Pisa, Italy), the Institute for Language and Speech Processing – Research and Innovation Center in Information, Communication and Knowledge Technologies (Athens, Greece), the University of Cambridge (Cambridge, Great Britain), Linguatec LG, (Munich, Germany), the Dublin City University (Dublin, Ireland), and ELDA (Paris, France).

PANACEA aims to reduce the language barriers in the European Union and tackles one of the most critical aspects of machine translation: the so-called language-resource bottleneck. Modern machine translation systems are data-driven which means that they require large mono- and multilingual corpora for training purposes for every language pair and each domain for which they are developed. These need to be updated regularly as language evolves constantly.

In order to equip language technology developers with up-to-date corpora for all languages spoken in the European Union, PANACEA's goal is to create a platform that automates acquisition, production, updating and maintenance of language resources required by machine translation systems and and other language technologies. This project is planned to run for three years and has started in 2010. It is split into seven work packages where the most relevant for this work are 'The PANACEA Platform' (WP3) and 'Parallel Corpus and Derivatives' (WP5).

1.3.2 ELDA

I have done this work as a member of the PANACEA participant ELDA (*Evaluations and Language resources Distribution Agency*). The company is the operational body of ELRA (*European Language Resources Association*) that has been set up by the European Commission and a number of European universities in 1995 in order to promote language resources for the Human Language Technology (HLT) sector, and to evaluate language engineering technologies.

[1] PANACEA: http://www.panacea-lr.eu

ELDA has been established to collect and produce language resources that are useful for the HLT community.

The developments presented in this document are intended to be used in the PANACEA work packages ELDA is involved in as far as applicable, but the work presented in this document has been done independently. However, some decisions have been influenced by the PANACEA project's needs. These decisions are clearly indicated in the text; all others are made by the author of this dissertation. Regarding the practical parts of this work, PANACEA has not had any influence because at the time of writing this work, the relevant work packages have not yet entered the phase of implementation.

1.4 Related Work

The following projects have goals similar to those of this work and those of PANACEA. Their approaches differ but I have studied their architectures as potential inspirations. For PANACEA, they are considered to serve as potential fall-back solutions.

1.4.1 GATE

The General Architecture for Text Engineering[2] (Cunningham et al., 2001) aims to support all kinds of natural language processing tasks. It provides four core components: an integrated development environment (GATE Developer), a web-based collaborative annotation environment (GATE Teamware), an object programming library (GATE Embedded) and a process for the integration of data and models (GATE Process).

Additional components provide further GATE functionalities: GATE Mímir is an indexing and repository management tool, GATE Wiki is a collaboratively usable content management system, and GATE Cloud is a parallel distributed engine that runs GATE Embedded in super computers.

1.4.2 UIMA

The Apache UIMA[3] (Unstructured Information Management Architecture) project (Ferrucci and Lally, 2004) is a general approach to support processing of unstructured data; including text, audio, speech, images, and video. UIMA components can be written in Java or C++ and are integrated into the UIMA framework using XML descriptions.

UIMA's original goal was to make the cooperation between the IBM research centres faster and more efficient. Meanwhile, it has become an Open Source project hosted by the Apache Software Foundation.

[2]GATE: http://gate.ac.uk
[3]Apache UIMA: http://uima.apache.org

1.5 Terminology

This section describes terminology used in this document.

- NLP: Natural Language Processing is the scientific area in which computers deal with natural languages. Usually, the term refers to written language (texts) and addresses tasks involving natural language understanding.
- Corpus: a set of documents in one or multiple languages (monolingual or multilingual corpus respectively).
- Document: a self-contained text that usually provides information about one topic.
- Parallel Corpus: a multi-lingual corpus containing texts that are organised as pairs such that each document in one language is assigned to its translation(s) in the other language(s).
- Parallel Sentences: sentence pairs in a parallel corpus that are translations of each other.
- Word-/Chunk-/Sub-tree-Aligned Sentences: a parallel sentence pair in which the basic sentence units (words, chunks, or sub-trees) of the sentence in one language are assigned to their corresponding parts in the translated sentence.
- Hansard: a transcript of the discussions in the parliaments of the United Kingdom and of most members of the Commonwealth of Nations. In Canada, Hansards are written in both official languages, English and French.
- Scripting Language: a programming language that is used to write scripts. Scripting languages often use interpreters rather than compilers, meaning that the source code's compilation into machine language is performed during the programme execution. The scripting languages that are used in this work are Perl[4], Bash[5], and C-Shell.
- Script: a usually not too complex software programme written in a scripting language.
- Wrapper: a software programme that calls another programme, e.g. in order to adapt the format of the input parameters to specific needs.
- Unix Standard Streams: a Unix-based operating system provides three streams for every programme that is run from its command line: the programme may read input from standard input (*stdin*), print results to standard output (*stdout*), and print diagnostic messages to standard error (*stderr*).
- HTTP: the Hypertext Transfer Protocol is an object-oriented, application-based protocol used by the World Wide Web. It 'allows an open-ended set of methods that indicate the purpose of a request' (Fielding et al., 1999).

[4]Perl: http://www.perl.org
[5]Bash: http://www.gnu.org/software/bash

- URI: a Uniform Resource Identifier is a strictly defined string that uniquely identifies abstract or physical resources such as web pages, files, web services, e-mail addresses and addressees, etc.
- URL: a Uniform Resource Locator is a specific type of URI that describes a network resource by defining the transport protocol and its location.
- XML: the Extensible Markup Language is a markup language for the text-based representation of structured data.
- HTML: the Hypertext Markup Language is a markup language that describes the contents of (mainly) Word Wide Web documents such as texts, images, hyperlinks, and meta-information.

These are the different font styles used in this document:

- *Italic*: Text segments written in *italic* letters reflect technical terms, proper names, e.g. of a software programme, texts in languages other than English, or variables.
- `Typewriter (fixed width)`: A text segment written in `typewriter` style refers to a command or programming language segment and shows a command or source code in the exact spelling that is used as input.

Chapter 2

Alignment

2.1 Introduction

When speaking about alignment, one has to specify which kind of alignment is meant. In the context of this work, alignment describes the task of identifying corresponding parts in two corpora. In the platform that is developed here, the term refers to texts – documents or sentences – that are translations of each other (parallel corpus). In general, a sentence aligner could also treat texts that cover the same topic in different languages without being exact translations (comparable corpus); in that case, aligning sentences is mainly a task of extracting parallel sentences, i.e. translated sentence pairs. In order to specify the alignment type, the size of the aligned units can be used (paragraphs, sentences, word, phrases, etc.).

Considering one or more parallel documents, alignment is usually performed on the sentence level at first (see section 2.2), although this can be preceded by segmenting the texts into larger units like paragraphs. Then, for each sentence in one language, its counterpart(s) in the translation need to be identified. Descending to a level of smaller units, the parts of aligned sentence pairs – parallel sentences – are aligned with each other (see section 2.3).

The term alignment is used in the field of phonetics as well. In phonetic alignment, phonemes are aligned with certain segments of recorded speech in order to prepare data for use in technologies like speech recognition and speech synthesis. This topic will not be covered in this document.

2.2 Sentence Alignment

> The sentence alignment task is to identify correspondences between sentences in one language and sentences in the other language. (Gale and Church, 1994)

Making parallel corpora production out of multilingual texts more efficient leads to a larger amount of available data. This means that NLP applications that exploit this kind of data e.g. with machine learning techniques, will profit. This concerns statistical and example-based machine translation systems, but other technologies such as rule-based transfer grammar induction

and development depend on large corpora, too. While bilingual corpora are available for many languages, e.g. the protocols of multi-lingual parliaments like in the European Union and in Canada, these transcriptions usually do not mark parallel sentences.

2.2.1 Length-based Approaches

In (Gale and Church, 1994), the sentence alignment is prepared by aligning the paragraphs with each other first. This is a 'fairly easy' (Gale and Church, 1994) process for the corpus in use: a trilingual corpus (English, French, and German) by the Union Bank of Switzerland (UBS) with 725 sentences and 188 paragraphs in the English text. It provides paragraph boundary markers that are exploited to segment the text into sections of well manageable size. So-called pseudo-paragraphs like headings and signatures are removed by applying length thresholds because they were not always translated. During the manual check of the results, the authors also discovered mistakes produced during the translation, e.g. in one document a paragraph was omitted while another one was duplicated. Documents with errors of this kind were removed.

In the second step, the sentence alignment task is tackled by a simple statistical approach based on sentence lengths, assuming that a long sentence in one language corresponds to a long sentence in the translation. A probability is assigned to each possible sentence pair based on the ratio of lengths in characters and the variance of this ratio.

The most likely of all possible combinations is then computed assuming a Gaussian normal distribution with the mean being based on the expected number of characters in one language for each character in the other. As the naïve approach – computing every possible combination – has an exponential computational complexity, a dynamic programming approach is used to efficiently find the best alignment as in most sentence alignment algorithms.

For 1:1-alignments, i.e. one sentence in one language corresponds to exactly one in the other language, error rates of 2.6% for English-French and 1.4% for English-German have been achieved. The test corpus contained 542 out of 620 (87.4%) 1:1-alignments in the English-French part and 625 out of 695 (89.9%) in the English-German one (Gale and Church, 1994). However, there are less trivial cases than 1:1 sentence alignments:

> At times, a single sentence in one language is translated as two or more sentences in the other language. At other times a sentence, or even a whole passage, may be missing from one or the other of the corpora. (Brown, Lai, and Mercer, 1991)

The statistical model presented by (Gale and Church, 1994) suffers from much higher error rates for alignment categories other than 1:1: 9% for 2:1 and to 33% for 2:2 alignments (with 117 and 15 occurrences in the test corpus respectively). In all other alignment categories, the algorithms has failed completely (100% error rate for the 31 occurrences of 3:1 and 3:2 alignments).

(Brown, Lai, and Mercer, 1991) present a similar approach, but measure sentence lengths based on the number of words instead of characters. The method relies on a specific property of the English-French Hansard of the Canadian parliament: it contains comments providing

information such as speaker names and temporal information that are exploited as markers in the text, called anchors. The resulting segments can be used in a similar way as the paragraph boundaries in (Gale and Church, 1994) but with the advantage of the anchors being either unique (temporal information) or at least infrequent (speaker information). The model allows for 1:0, 1:1, and 1:2 sentence alignment and achieves a total error rate of 0.9%.

The length-based approaches work well for most cases but depend on certain specifics such as the presence of comments (Brown, Lai, and Mercer, 1991) or paragraph markers (Gale and Church, 1994). Neither of them can handle bad translations; mistakes like missing and doubled passages have to be dealt with manually.

2.2.2 Approaches Based on Lexical Information

Approaches based on lexical information have been proposed aiming mainly at increased robustness for corpora that contain deletions and other erroneous transcriptions. The methods presented by (Kay and Röscheisen, 1993) and (Chen, 1993) are based on the creation of word-to-word correspondences that are computed iteratively by their respective similarities in distribution. Supported by this simple word level alignment method, the sentence alignment is derived subsequently by assigning scores according to word correspondences in potentially parallel sentences. However, the approach presented by (Kay and Röscheisen, 1993) is computationally too expensive to be applied on big corpora.

(Chen, 1993) creates a simple statistical word-to-word translation table during the sentence alignment process. The method requires an initial seed of manually aligned sentence pairs. A variant of the Expectation Maximization algorithm is applied in order to retrieve lexical information (Dempster et al., 1977). It uses the initial seed set as a starting point and then iterates over the whole corpus. The lexical information gained through this process allows an exact sentence alignment and the identification of passages that have been deleted in one of the translations.

An error rate of only 0.4% after an initial manual alignment of 100 sentences shows improvement compared to the length-based methods, even with erroneous translations. The price for the increase in quality and robustness is, besides the required manual alignment for the seed set, that the algorithm is 'tens of times slower than the Brown and Gale algorithms' (Chen, 1993).

2.2.3 State of the Art

Most research on sentence alignment is based on the approaches presented in the preceding sections and tries to optimise and combine them. In (Melamed, 1996; Melamed, 1997; Simard and Plamondon, 1998), bitext correspondence maps between the two translations of a text are created, i.e. trying to find text segments that indicate semantic correspondence.

In order to create a network of aligned words, the Smooth Injective Map Recognizer (SIMR) algorithm presented by (Melamed, 1996) uses simple heuristics to find cognates – words with a similar etymology – by detecting long common subsequences in the respective orthographies of

words in translations. (Simard and Plamondon, 1998) identify possible translations by searching for words that begin with the same four letters. They are identified as 'isolated cognates' if no resembling words occur within an isolation window whose size depends on the length of the given word. (Melamed, 1996) can additionally apply a bilingual dictionary, if available, to create word mappings.

In either approach, words that are identified as correspondences are used as initial bitext mapping points that are then exploited to generate the most likely sentence alignments. With an error rate of 0.48%, SMIR achieves better results than the length-based methods without the high computational complexity required by previous lexical-based approaches (Kay and Röscheisen, 1993; Chen, 1993). Its downside is that it is limited to language pairs for that either a dictionary is available or in which both languages have a high amount of cognates that can be detected by analysing the orthography.

In the search for a method that is 'fast, highly accurate, and requires no special knowledge about the corpus or the two languages', (Moore, 2002) presents an approach that works without an external dictionary, with any language pair, and without specific requirements such as paragraph boundary marks or anchors.

In the first step, a model based on sentence lengths is applied like in (Brown, Lai, and Mercer, 1991) in order to produce a basic alignment. Subsequently, a basic word alignment is created using a variant of the IBM Translation Model 1 (Brown et al., 1993) (see section 2.3.1) with the limitation that it is only applied to sentences initially aligned with a very high probability (0.99). In order to increase both robustness and efficiency, translations of rare words (two or fewer occurrences) are omitted. In a third step, the initial sentence alignment is modified using the word alignments yielding the final sentence alignment.

(Moore, 2002) considers that 'it is sufficient [...] to extract the 1-to-1 alignments' as they are 'in practice the only alignments that are currently used for training machine translation systems.' For this limited focus, the results both precision and recall lie above 99% for any parameter settings. With 300 sentences deleted randomly in one of the translations to test the algorithm's robustness, 'the precision error is 13.0 times lower and the recall error is 37.4 times lower than with the sentence-length-only-based method' with a modest increase in computational cost.

(Varga et al., 2005) present the *Hunalign* algorithm (see section 4.1.2). Its strategy is to exploit a bilingual dictionary if available in order to optimise results, but also to perform without the assistance of external resources. At first, 'a crude translation of the source text is produced by converting each word token into the dictionary translation that has the highest frequency in the target corpus' (Varga et al., 2005). If no translation is available in the dictionary, the original word is used as its own provisional pseudo-translation. The latter is the case for all words if there is no dictionary available for the given language pair. This implies that only words that are the same in both languages, including proper names etc. are correct in the pseudo-translation.

Subsequently, *Hunalign* computes a similarity score between the sentences in the pseudo-translation and the actual translation out of two factors. A token-based component counts the

number of shared words and normalizes by the number of tokens in the longer sentence. The second scoring component is the correlation in length based on the number of characters in the two sentences. In conclusion, the algorithm applies the combination of dictionary-based and length-based approaches similar to (Moore, 2002) but with the additional option of falling back to a purely length-based approach if necessary.

The evaluation presented by (Varga et al., 2005) compares *Hunalign* to the approach by (Moore, 2002). When considering 1:1 sentence alignments only, as in (Moore, 2002), Hunalign achieves for both recall and precision values of 99.4% for the English-Hungarian translation of George Orwell's *1984* with the use of a dictionary. When including one-to-many-alignments into the evaluation, both precision and recall remain as high as 99.3%. Additionally, *Hunalign*, implemented in C++, is 'at least an order faster than Moore's implementation (written in Perl)' (Varga et al., 2005).

2.2.4 Different Transcriptions

The length measures presented so far in this work were designed for languages using the same alphabets. (Wu, 1994) demonstrates that the length-based approach counting characters (Gale and Church, 1994) works for aligning English-Chinese bilingual corpora too, as tested on the transcriptions of the Hong Kong parliament Hansard. In total, 86.4% of the alignments have been identified correctly; when ignoring the introductory session header with its domain-specific formats, the length-based approach correctly identifies 95.2% of the alignments which is close to the English-French rate of 96-98%. However, measuring sentence length by counting words as in (Brown, Lai, and Mercer, 1991) is not easily feasible as Chinese word segmentation had to be applied first, potentially yielding additional errors.

(Wu, 1994) furthermore applies a very limited dictionary. In order to ensure that all the translations in it are correct and unambiguous, a highly domain-specific lexicon has been created manually that contains very few entries such as the months and very corpus-specific terms such as 'C.B.E' (*Commander of the British Empire*). This improves the algorithm's results for the introductory sentence headers so that the overall success rate raises to 92.1% while the rate for the corpus with the headers being omitted does not change significantly due to the dictionary.

The alignment algorithm *Champollion* (Ma, 2006) applies dictionary translations and aims to improve alignment quality by assigning weights to them. By applying a *tf-idf*[1]-like weighting scheme, the *segment-wide term frequency* (*stf*), rare words are considered to be more relevant for aligning sentences. In order to find the most likely alignment for a bilingual document, Champollion uses *stf* to compute a similarity score for the alignment candidates while penalizing differences in lengths and any alignment type other than 1:1. Champollion requires a dictionary, but is language-independent apart from that. Additionally, a tokenizer, a word segmenter for Chinese, and stemmers for morphological languages, including English and Arabic, are applied in order to prepare the alignment process.

[1]Term Frequency-Inverse Document Frequency: a statistical measure to determine a word's importance in a document in relation to a corpus

The results presented in (Ma, 2006) show that the outcome quality is correlated with the dictionary size. Without a lexicon, i.e. only translating words that are the same in both languages such as proper nouns, it achieves a precision and recall of 88.1% and 90.8% respectively. These rates raise to 97.0% and 96.9% with a 58,000 words dictionary, allowing for n:m-alignments with n and m from 0 to 4. The approach's advantage is that it deals well with noisy inputs and different alignment types as well as with any language if lexical resources are provided.

2.3 Sub-sentence Alignment

The alignment task on a sub-sentential level is to assign the basic units of a sentence in one language to their corresponding parts in the translation of that sentence. The simplest units that can be aligned are words (see section 2.3.1). From words, phrases can be derived, too; these are groups of words that form regularly used expressions, but are not necessarily grammatical units. In the case of words, aligning two words in different languages with each other can be interpreted as defining them as possible translations of each other.

The basic units of a sentence can be larger than single words. A more linguistically motivated approach aims to align chunks from parallel sentences (see section 2.3.2). A chunk is a syntactical constituent, e.g. a noun phrase or a verb phrase, i.e. it can consist of one or multiple words. Subtree alignment (see section 2.3.3) takes the syntax trees of the parallel sentences into account and aligns partial trees (sub-trees) with each other.

2.3.1 Word-based Alignment

The IBM Models 1-5 presented by (Brown et al., 1993) form a generative approach for word-by-word alignment. Their common basic assumption is that any string e in one language can be generated by any string f in the other language. Ideally, the model is estimated so that the conditional probability $P(f|e)$ is highest for the best translation. Following the noisy-channel model (Weaver, 1955), 'we further take the view that when a native speaker of French produces a string of French words he has actually conceived of a string of English words, which he translated mentally' (Brown et al., 1993).

According to this idea, the model aims to estimate an alignment a that maps all the strings in a sentence in one language to those from which they have been generated, i.e. the matching strings in the other language. An English text passage is seen as 'a web of concepts woven together according to the rules of English grammar' (Brown et al., 1993).

The visible part of these concepts are the words: 'we cannot see the concepts directly, but only the words that they leave behind' (Brown et al., 1993). These units are called *cepts*. Each word can be part of none, one, or multiple cepts and a cept can consist of several words. During the translation into another language, the cept generates the foreign words and the alignment task aims to find out which words are part of which cepts, the *ceptual scheme*. Words in the foreign language that have no counterpart in the original language are generated by the empty cept.

For each sentence pair (e, f) exists a set of possible alignments $A(e, f)$. With l and m being the lengths of the two sentences e and f respectively, the size of $A(e, f)$ is 2^{lm}. The optimal alignment a maximises $p(e, a|f)$ for the sentence pair (e, f). The basic idea that underlies all the IBM Models is to apply the Expectation Maximization algorithm (Baum, 1972; Dempster et al., 1977), and to find the most likely alignments based on common occurrences in the parallel corpus.

In Model 1, all alignments in a sentence pair are initially assumed to be equally likely. After a first run over the parallel corpus, the translation probabilities $t(f|e)$ are adapted based on the number of co-occurrences of the string pair (e, f) according to equation 2.1 with J and I being the sentence lengths and i and j the word positions. If there is no sentence pair containing both the string e in a sentence and the string f in its translations, $t(f|e)$ results in 0. The procedure is repeated iteratively with the newly estimated translation probabilities until the values converge. It eventually yields a translation table containing alignment probabilities and the alignment eventually is chosen that maximises the overall probability according to this table.

$$Pr(f_1^J|e_1^I) = p(J|I) \prod_{j=1}^{J} \sum_{i=1}^{I} [p(i|j, l) \cdot p(f_j|c_i)] \tag{2.1}$$

IBM Model 2 is an extension of Model 1 which aims to achieve better alignments by setting two preconditions: 'the probability of a connection depends on the positions it connects and on the length of the two strings' (Brown et al., 1993). These presumptions are implemented by using the conditional probability $a(i|j, m, l)$ as an additional factor when computing the translation probabilities $t(f|e)$; here, i is the word position in the source language, j the word position in the target language and m the length of the target string.

IBM Models 3, 4, and 5 choose for each word in a string the number of corresponding words in the other language before searching the words themselves. This introduces the idea of fertility:

> In the pair (*Jean n'aime personne*|*John loves nobody*), we can align *John* with *Jean* and *loves* with *aime*, but now, *nobody* aligns with both *n'* and *personne*. Sometimes words in the English sentence of the pair align with nothing in the French sentence, and similarly, occasionally words in the French member of the pair do not appear to go with any of the words in the English sentence. [...] We call the number of French words that an English word produces in a given alignment its *fertility* in that alignment (Brown et al., 1990).

This idea aims to improve alignment quality by assigning a higher likelihood e.g. for the English word *cheap* to be aligned with two French words (*bon marché*) rather than with one or three. In IBM Model 3, after the fertility model has estimated the number of foreign words to align with an English word, the alignment probabilities are computed on the base of the positions and string lengths like in Model 2. Additionally, Model 3 introduces a distortion model that estimates the probability of a word to appear on a position in the target sentence that is different from the position in the source sentence.

Model 4 adds another component to Model 3 that considers the movement of phrases instead of single words only. This is motivated by the linguistic insight that movements are considered to be less likely in case of long phrases in comparison to short phrases which is why Model 4 does not penalise moving phrases. In opposition to that, every word of a moving phrase decreases probability values in Model 3.

Both Model 3 and 4 are *deficient*, meaning that they are 'not concentrating all of [their] probability on events of interest' (Brown et al., 1990). This refers to the fact that the distortion model introduced in Model 3 assigns positions to words independently of the previous words, i.e. 'Model 3 wastes some of its probability.' In practice, this leads to more and more words being aligned with the empty word during the EM iterations (Och and Ney, 2003). Ignoring deficiency simplifies the alignment computation, but in Model 4, where phrases can move as a whole, this allows for words to lie on top of one another and to be placed before the first or beyond the last position. Model 5, instead, makes sure that both single- and multi-word-cepts are placed into vacant positions.

In the noisy-channel model, a sentence's most likely translation is computed from two factors: the translation probability and the language model. The IBM Models are applied to compute the translation probabilities (Brown et al., 1990). The language model applied in (Brown et al., 1990) is n-gram based and computes the likelihood of a possible translation to occur in the target language.

The search for the most likely translation considering both translation and language model probabilities – while every combination of words is seen as a possible translation – 'face[s] the difficulty that there are simply too many sentences to try' (Brown et al., 1990). Therefore, a stack search (Bahl, Jelinek, and Mercer, 1990) is applied instead, omitting less likely partial translations at intermediate stages. However, this suboptimal method can fail to find the most likely translation as it might be omitted in an early stage. Therefore, modern machine translation systems like Moses (Koehn et al., 2007) use dynamic programming search algorithms instead that always yield the optimal result.

The IBM Models ignore the fact that 'the words are not distributed arbitrarily over the sentence positions' (Vogel, Ney, and Tillmann, 1996). Instead, they tend to occur in the same neighbourhood in both languages because movement is mostly performed in clusters. Therefore, (Vogel, Ney, and Tillmann, 1996) add a dependence for an alignment a_j on the previous alignment: $p(a_j|a_{j-1}, I)$ where I is the translated sentence's length which is introduced for normalisation.

The context-sensitive approach is realised with a Hidden Markov Model (HMM) where the alignments $a_1^J := a_1, a_j, \ldots, a^J$ for a sentence pair (f_1^J, e_1^I) are seen as the hidden variables. Additionally, the HMM alignment probabilities are assumed to depend on the difference between i and i' – the initial position and the mapped position – rather than on absolute positions. The full HMM alignment probability is presented in equation 2.2. It yields equation 2.3 as presented by (Och and Ney, 2000) which makes it comparable to the IBM Model 1 formula (see equation 2.1).

$$p(i|i', I) = \frac{s(i - i')}{\sum_{l=1}^{I} s(l - i')} \tag{2.2}$$

$$p(f_1^J|e_1^I) = \sum_{a_1^J} \prod_{j=1}^{J} [p(a_j|a_{j-1}, I) \cdot p(f_j|e_{a_j}) \tag{2.3}$$

As a further improvement, (Och et al., 1999) proposes the usage of *alignment templates*. The idea is to describe sequences by their word classes so that a sequence containing e.g. a certain town name, can be applied for all sequences that contain any town name at that position. Formally, an alignment template describes the alignment $\tilde{A}$ between two sequence classes. $\tilde{A}$ is represented as a matrix that contains 1 at positions that are aligned with each other and 0 everywhere else. The sequence classes are produced from bilingual word classes that are learned automatically (Och, 1999).

(Och and Ney, 2000) propose to extend a training corpus with translations taken from an existing bilingual dictionary. It is applied during the training phase with the Expectation Maximization algorithm to improve alignments for both words in the dictionary and indirectly for other words too. An additional factor F_{lex} is introduced for the EM training which assigns a high weight to dictionary entries that occur in the training corpus and a low weight to all others. If, for instance, the weight of F_{lex} for a dictionary entry is 10 as in the experiments presented in (Och and Ney, 2000), that entry is added to the training corpus 10 times.

In (Och and Ney, 2003), the alignment templates and the external dictionaries are combined with the methods of IBM Model 5 from (Brown et al., 1993) to construct a new Model 6, implemented in the tool *GIZA++* (see section 4.1.2). Additionally, smoothing is applied for all models to reduce overfitting and improve the dealing with rare words. In order to allow for multi-word alignments in both directions, the training is performed with both translations being used as source as well as target language (symmetrisation).

In order to compare the IBM Models 1-5, Model 6, and the HMM-based alignment model using the heuristics shown above, (Och and Ney, 2003) evaluate these by using the German-English data from the Verbmobil task (Wahlster, 2000) and Canadian Hansards (French-English) for training. A subset of both corpora was aligned manually to produce a test corpus. The quality is measured by computing the F-measure (harmonic mean) of precision and recall for each model's results against the manually created gold standard.

In summary, (Och and Ney, 2003) shows that statistical methods outperform heuristic approaches. Model 6 outperforms each of the other models tested and both smoothing and symmetrisation improve every model's alignment quality. On the other hand, using a bilingual dictionary and considering word classes as templates does not yield significant improvements.

However, for the IBM Models 'it is impossible to learn that the phrase "b c" in a language S means the same thing as word "x" in language T' (Marcu and Wong, 2002). Therefore, they propose an approach in which a bag of concepts is generated and these concepts are represented by phrases consisting of one or multiple words. For each concept, a pair (e_i, f_i) is generated where e_i and f_i are phrases. In the Model 1 presented by (Marcu and Wong, 2002), the most

likely alignment is searched assuming a uniform distribution like in IBM Model 1. Model 2 in (Marcu and Wong, 2002) works similarly as IBM Model 4, by implementing a position-based distortion model, but on a phrase instead of a word level.

For the evaluation, the presented model was trained on 100,000 parallel French-English sentences from the Canadian Hansard corpus. Afterwards, it was used to translate 500 unseen sentences. In comparison with IBM Model 4 trained and tested on the same corpora, the phrase-based approach achieved a higher number of perfect translations (28% vs. 22%) and a higher average BLEU score (Papineni et al., 2002): 0.2325 vs. 0.2158.

2.3.2 Chunk Alignment

This alignment approach considers syntactic constituents such as noun phrases or verb phrases instead of words as the smallest units of a sentence. 'These chunks correspond in some way to prosodic patterns. [...] The typical chunk consists of a single content word surrounded by a constellation of function words, matching a fixed template' (Abney, 1991).

An assumption that makes chunks interesting for alignment purposes is that the number of chunks will be approximately the same in translated sentences and that most reordering is realised within the syntactic constituents. This approach implies that chunks need to be identified before the actual alignment tasks. Instead of a full syntax parse, shallow parsing can be applied, i.e. the identification of the syntactic constituents without analysing their internal structure or their role in the sentence. This is advantageous because syntax parsers lack robustness and 'do not well at identifying good phrases in noisy surroundings' (Abney, 1997). This is due to errors occurring in almost every text to a certain degree and to each parser's 'unavoidable incompleteness of lexicon and grammar' (Abney, 1997).

Function words or stop words can be used as a simple method to identify chunks. (Bourigault, 1992) uses this method to find noun phrases in French texts after part-of-speech (POS) tagging a text. There are grammatical patterns made up by elements like conjugated verbs, pronouns, conjunctions, and certain strings containing prepositions and determiners. These patterns provide 'negative knowledge' (Bourigault, 1992) about text passages that cannot be noun phrases. Subsequently, patterns consisting of part-of-speech tags are used to find noun phrases in the remaining text sequences.

(Church, 1988) presents a stochastic method to identify noun phrases in a POS tagged text. In this approach, POS tag sequences are assigned to probabilities using a HMM that computes each sequence's likelihood of starting or ending a noun phrase. The HMM training is performed on a corpus of 40,000 words in which 11,000 noun phrases were marked semi-automatically. As a result, the model estimates e.g. for the sequence (VB, AT) (a verb followed by an article) a probability of 1.0 that a noun phrase follows beginning with this article. The sequence (NN, AT) (a noun followed by an article), however, indicates with a probability of 1.0 that a noun phrase ends. Other sequences turn out to be more ambiguous.

The noun phrase marking is realised by a parser that inserts brackets indicating possible beginnings and endings of noun phrases into a sequence at all possible positions while noun

phrases cannot overlap. The possible bracket configurations are scored according to the probabilities estimated from the training corpus to find the most likely one. The authors do not present a formal evaluation of their approach, but have found a 'tendency to underestimate the number of brackets' (Church, 1988). The method missed only 5 out of 243 manually checked noun phrases.

The bracketing method is also applied by (Ramshaw and Marcus, 1995) but using rule-learning methods inspired by (Brill, 1994) instead of an HMM. (Skut and Brants, 1998) extend the Markov-Model-based bracketing approach by including a recogniser for prenomial adjectival and participal phrases, postnominal prepositional phrases, and genitives in order to identify not only noun phrases but also prepositional phrases and adjective phrases. In (Molina and Pla, 2002), the chunking task is treated as a tagging problem so that a HMM-based tagger can be applied for the identification of all kinds of chunks.

Alternatively, chunks are identified according to the linguistically motivated marker hypothesis (Green, 1979). It states that languages tend to use closed sets of lexemes and morphemes in order to achieve a syntactic structure on a surface level. (Gough and Way, 2004b) apply this hypothesis for the alignment part of a machine translation system. After manually defining six sets of marker words for different syntactic kinds of chunks for English and French, these markers are used to identify chunks in both languages: 'A new fragment begins where a marked word is encountered and ends at the occurrence of the next marker word. [...] Each chunk must contain at least one non-marker word' (Gough and Way, 2004b).

Earlier machine translation systems based on the marker hypothesis (Gough, Way, and Hearne, 2002; Gough and Way, 2003; Way and Gough, 2003) presume that parallel sentences will have the same number of chunks and that their order does not need to be changed, allowing for 1:1 chunk alignments only. In (Gough and Way, 2004a), lexical similarities and cognates are considered during the alignment process in order to match the chunks with their respective translations. Reordering is allowed, but penalised such that higher distances between chunks in a parallel sentence pair yield lower probabilities for matching each other. Additionally, n:1 alignments are introduced.

(Stroppa and Way, 2006) apply chunking based on the marker hypothesis to English and Italian texts. The presented system (MaTrEx) is supposed to work with Arabic too, where 'determiners, prepositions, and pronouns do not usually form independent tokens but are usually part of a token which also contains a noun, an adjective, or a verb' (Stroppa and Way, 2006). This implies that pre-processing of the corpus with tokenisation and POS tagging becomes necessary. However, introducing these complex tasks would cause the loss of one important advantage, simplicity, of which chunking based on the marker hypothesis generally profits. Therefore, (Stroppa and Way, 2006) do Arabic chunk detection using the Support Vector Machine learning method with training performed on the 4519 syntactically marked sentences provided by the Penn Arabic Treebank (Diab, Hacioglu, and Jurafsky, 2004; Maamouri et al., 2004).

2.3.3 Sub-Tree Alignment

A further step towards the consideration of syntactic context is taken in tree-based approaches where a full syntactic analysis of parallel sentences is performed in a first step. Afterwards, sub-trees of the resulting syntax trees are aligned with each other.

Generally, the approach is motivated by the idea that a translation depends on its context, so the information provided by a syntax tree should improve results. In (Samuelsson and Volk, 2006), a German-English-Swedish parallel treebank has been created based on the novel 'Sophie's world'. After automatically creating mono-lingual treebanks for each of the languages by using syntax parsers, the alignment has been done manually. This work demonstrates one central difficulty of the task: even the three human aligners disagree in up to 11% of the cases, creating one unique Gold standard for evaluation purposes is therefore impossible.

In order to make multi-lingual treebanks useful for applications such as data-driven machine translation (Hearne et al., 2007) and grammar induction (Nesson, Shieber, and Rush, 2006), their production needs to be automatised. (Zhechev and Way, 2008) present a sub-tree aligner for the automatic generation of parallel treebanks. It requires word alignment tools such as GIZA++ for both involved languages and makes use of a syntactic parser if available; POS taggers can be used as a fall-back option, but are not recommended.

In a first step, the the sub-tree aligner maps parallel texts on a word level. If parsers are available, monolingual syntax trees are created in the second step. Then, the algorithm computes equivalence scores to create links between the nodes in both trees. These scores are based on the word-alignment probabilities for the single words of a sub-tree pair and on the strings surrounding them. If no syntactic parser is available for one or neither of the languages, the sub-tree aligner falls back to tree-to-string or string-to-string alignment where trees are created as well, but with all the nodes having the same label.

(Chiang et al., 2005) shows that the syntax does not need to be linguistically motivated, but can also be generated automatically as in the *Hiero* system. It 'extends the standard non-hierarchical notion of "phrases" to include nonterminal symbols' (Chiang et al., 2005) where nonterminals represent nodes in the syntax tree while the leaves are terminals, i.e. the actual words. The model's syntax uses a context-free grammar (CFG) without any linguistic analysis. Therefore it works without any syntactically tagged training data. Compared to the phrase-based machine translation system Pharao (Koehn, 2004), Hiero improves the BLEU score from 0.2676 to 0.2877 for English-Chinese translations (Chiang, 2005).

Chapter 3

Web Services

3.1 Introduction

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format. (Haas and Brown, 2004)

From a more practical point of view, a Web service provides software and data from a web server to web users and web-connected programmes. This means that the offered services can be dynamically included into any programme running on a computer that is able to connect to the server. A client can send variable data to a web service, if required, and retrieve results that depend on this input. That way, web services can be part of a distributed system that executes different computations on different machines and collects the results from the involved services.

3.2 Describing a Web Service with WSDL

A Web service is described in a dedicated XML-based language, WSDL (*Web Services Description Language*). It provides information about its functionality and how to access it to potential clients. This includes the definition of the interface, the protocol, and the available functions. However, there is no semantic information like the purpose of a function's arguments and return values included. Appendix A.1 shows an extract of the WSDL file that describes the Hunalign Web service generated with Soaplab (see section 4.1.2).

In the WSDL paradigm, abstract descriptions are used to create concrete messages that are sent to human and machine users by wither applying existing transport techniques such as HTTP (Fielding et al., 1999) directly or SOAP as an additional layer (see section 3.2.2).

3.2.1 Abstract WSDL Elements

To define a service's capabilities, WSDL uses four abstract element types. This means that they do not give information about how to actually access the service via the network but rather outline its communication abilities.

- *types*: defines data types that can be used as input and output. The types that are predefined in the XML Schema Definition Language (XSD) (Peterson et al., 2009) are sufficient for many cases as common primitive types like *string*, *integer* etc. are provided and complex types can be defined in addition.

- *message*: comprises one or multiple *part* entries that represent the data the message contains, i.e. a variable's name and type. Their names are set with the *name* attribute. Message parameters are defined as *parts*. For each part, there is a *name* and a *type* attribute referring to one of the previously defined *types*.

- *portType* (in WSDL 2.0: *interface*): defines 'a collection of operations that are collectively supported by an end point' (Curbera et al., 2002), i.e. the service's available functions.

- *operation*: refers to an action a client can call on the server, i.e. execute a function, and defines the messages it expects and returns (`<input [...]>` and `<output [...]>` respectively) to perform that operation.

3.2.2 WSDL and SOAP

> SOAP is a lightweight protocol for exchanging structured information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses. (Gudgin et al., 2001)

In order to integrate SOAP (Simple Object Access Protocol) into a Web service description, WSDL uses the *binding* XML element. It has got a *name* and a *type* attribute that refers to a *portType* (see section 3.2.1). Within the *binding* environment, WSDL describes how to access the offered Web service concretely over the network. This includes the SOAP actions to be called, i.e. their URLs, the encoding standard, and the binding style. As an alternative to SOAP, WSDL can also directly use HTTP and MIME with respective bindings (Christensen et al., 2001).

The binding is assigned to its network address in the *service* element. It refers to a binding name using the *port* tag (in WSDL 2.0: *endpoint*). In there, the URI (Uniform Resource Identifier) is declared with the *soap:address* tag. An extract of the WSDL definition file automatically created by Soaplab is attached to this document in appendix A.1.

In a Web service using WSDL and SOAP, the task of the latter is to form an envelope around the WSDL messages and to send and receive WSDL messages via a network. SOAP generally allows the exchange of data between systems over a network and the execution of remote procedure calls. It uses available network protocols to transport messages, usually HTTP (Curbera et al., 2002), but transport protocols like FTP, SMTP, etc. are possible as well.

The communication involves three questions: '*what* communication protocol to use (such as SOAP over HTTP), *how* to accomplish individual service interactions over this protocol, and *where* to terminate communication (the network address)' (Curbera et al., 2002). The questions of *what* and *how* are addressed by the WSDL *binding* element and the *where* is addressed by the *port* element.

3.2.3 SOAP Implementations

There are several implementations for the application of the SOAP protocol. JAX-WS[1] (*Java API for XML Web Services*) is a Java API and by default uses SOAP messages and HTTP. As an alternative, Apache Axis[2] implements the SOAP protocol as well. Because both of these implementations comply well with the SOAP standard, the choice is mainly a matter of matching the client's capabilities rather than the respective protocol's properties.

3.3 REST

The Representational State Transfer (REST) (Fielding et al., 1999) allows for direct access to any kind of resource, including web services, directly using HTTP to transfer data rather than introducing an additional layer like SOAP. REST only describes methods of how to apply existing standards for distributed systems in the World Wide Web (WWW), but not a concrete standard or implementation.

In the REST paradigm, any WWW resource – like HTML pages, images, PDF files, etc. – is an object that can be retrieved and embedded in an application via its URI. Its content is what the server provides and the only condition for the successful communication is that server and client agree on the data format. For Web services, XML-based formats (WSDL) are recommendable for practical reasons as they are both human-readable and easy to parse. However, resources like HTML pages and binary data can also be accessed directly by a client.

REST applications use the HTTP protocol to transfer data. The data transfer from server to client is initiated with the HTTP `GET` command and the transfer from client to server with `POST`. In addition, the client can create and remove resources on the server with `PUT` and `DELETE`.

A 'restful' server, i.e. one that complies with the REST specifications, is stateless as it does not protocol the client's state. Therefore, the order of accesses to resources is entirely defined by the client. Authentication, if required, is done using the HTTP built-in authentication (Franks et al., 1999; Fielding et al., 1999).

3.4 Web Application Technologies

A web application allows a client to remotely execute a programme that is installed on a host. The aim of a *remote procedure call* (RPC) is usually to send user-specific arguments to the

[1] JAX-WS: https://jax-ws.dev.java.net
[2] Apache Axis: http://ws.apache.org/axis

executed function and to retrieve the according results. The ability to call software programmes in order to generate a user-specific output is an essential part of a Web service.

A main difference between a Web service and a web page is, regarding the output, the language that is generated. In the case of web pages, the client is usually a web browser that expects HTML pages while Web services are described in WSDL that are readable by appropriate clients. However, these languages are commonly used only because they serve the respective application requirements best (human reader with a web browser vs. machine client accessing a Web service), but are no technical requirements. There are different technologies available to create dynamic web sites of any kind, of which the most relevant are presented in the following sections.

3.4.1 The Common Gateway Interface (CGI)

In order to provide any web content, one requires a web server that allows and controls the network connections. To offer a Web service, the web server additionally needs the ability to execute an RPC rather than transferring static data from server to client only. One option to achieve this are CGI programmes (Common Gateway Interface):

> CGI is the part of the Web server that can communicate with other programs running on the server. With CGI, the Web server can call up a program, while passing user-specific data to the program (such as what host the user is connecting from, or input the user has supplied using HTML form syntax). The program then processes that data and the server passes the program's response back to the Web browser. (Gundavaram, 1996)

A CGI programme is usually developed similarly to an ordinary command line programme but implementing the interfaces that are defined by CGI. They allow the web server to execute the programme locally according to a client request and to retrieve its output. There exist programming libraries for the CGI implementation in most common programming languages.

A CGI server can take input as part of the URL separated by a `?`, for example:

```
http://example.com/translate.cgi?word=example
```

Multiple inputs can be concatenated each separated by another `?`. These and further information about the client like its IP address are stored as environment variables before calling the CGI programme. This allows the CGI to access the given information.

A CGI's output consists of two sections that are separated by a blank line. The first section comprises one line that defines the output format, e.g. `Content-Type: text/html` for HTML output. The second provides contains the actual content, i.e. generated HTML code or other data.

3.4.2 Java Servlets

The Java programming platform provides servlets (Davidson and Coward, 1999) as an alternative to CGIs:

> A servlet is a small pluggable extension to a server that enhances the server's functionality. [...] While servlets can be used to extend the functionality of any Java-enabled server, they are most often used to extend web servers, providing a powerful efficient replacement for CGI scripts. (Hunter and Crawford, 2001)

Servlets are realised as Java *javax.servlet* classes[3]. Like CGIs, this class itself does not provide any web server functionality, so servlets require a web server that supports the servlet technique: a *servlet runner* or *servlet container* to which a servlet can be deployed. A servlet container can operate in *standalone* mode (a web server providing servlet support) or as an *add-on* (an extension to a third-party web server) (Davidson and Coward, 1999).

Java servlets are platform independent: a servlet written on one operating system running a certain servlet container can be deployed on a different servlet container on a different system as well.

The most popular freely available Java servlet containers are Apache Tomcat[4] and Glassfish[5]. In this work, Tomcat has been chosen in this work because the combination of Soaplab (see section 4.1.1) with Tomcat is well documented. A comprehensive technical comparison between the available servlet containers has not been performed because web servers are not the focus of this work and by design, servlets are expected to be usable with any servlet container.

3.4.3 Comparing CGI and Java Servlets

The Common Gateway Interface was developed to allow the communication between a web server and other processes running on one host. This implies that the other programme is being started separately every time a client makes the web server execute it. In the case of a frequently visited site, this leads to CGI applications having 'perhaps the worst life cycle imaginable: When a server receives a request that accesses a CGI program, it must create a new process to run the CGI program and then pass to it, using environment variables and standard input, every bit of information that might be necessary to generate a response' (Hunter and Crawford, 2001). In addition, the server must wait for the CGI to finish without being able to interact in the meantime, so the CGI programme cannot, for instance, write output or error messages to the web server log files.

This behaviour is especially problematic for CGIs written in interpreted programming languages like Perl which is in wide use because of its text processing capabilities. For every execution of the programme, a separate Perl interpreter is loaded. However, this memory-intensive

[3] *javax.servlet* documentation: http://download.oracle.com/docs/cd/E17802_01/products/products/servlet/2.2/javadoc/javax/servlet/package-summary.html

[4] Apache Tomcat: http://tomcat.apache.org

[5] Glassfish: https://glassfish.dev.java.net

and computationally expensive operation can be avoided by using compiled programming languages such as C++ instead. The performance and memory issues have also been addressed by FastCGI (Brown, 1996) where a programme is loaded into memory only once and can then answer to multiple requests.

Java servlets have been designed to create an improved alternative, so the CGI drawbacks have been resolved. The price is that they always require a servlet container that most web servers do not include. Also, servlets completely depend on Java, while the CGI protocol can be implemented in any programming language, including compiled ones.

3.4.4 Other Technologies

There are other technologies that can be used to generate dynamic web pages and therefore for Web services, too. They are not discussed in detail in this work because they are not as easily applicable in this context; mainly because the development takes part within the PANACEA project where certain decisions like using Open Source software and the Linux operating system have been set as fixed preconditions. More generally, their potential advantages do not apply for this project and therefore do neither justify their costs nor the additional installation and configuration effort while Open Source software is available for free and easily installable on a Linux system. However, a brief overview of other available technologies is provided in this section.

Active Server Pages (ASP)[6], developed by Microsoft, became popular thanks to 'both its ease of programming and the wide-spread availability of the IIS server' (Sandvig, 2004). However, as described above, availability is not an issue in this project because a Tomcat server can be used effortless. Although ASP is available for the Apache server as well, its second advantage, the easily learnable programming language, is not considered as an argument here because the implementation itself has not been a problematic issue during this work.

Coldfusion[7] is a technology developed by Adobe Systems. It features an application server and a markup language: the ColdFusion Markup Language (CFML) aims to make the development of Web services more efficient by providing easily usable functions and tags that directly produce WSDL output. But easing implementation is not the focus here, so this server has not been considered to be used.

[6]Active Server Pages (ASP): http://msdn.microsoft.com/en-us/library/aa286483.aspx
[7]Adobe Coldfusion: http://www.adobe.com/products/coldfusion/

Chapter 4

Development of a Web-based Alignment Platform

This chapter describes the implementation of available tools for sentence and word alignment as Web services and their integration in such a way that a pipeline from an input corpus via a sentence-aligned corpus towards a word-aligned corpus can be created. This includes the implementation of scripts to wrap around the command line calls for the different tools and to convert input and output formats where necessary

For the sentence alignment, Hunalign 1.0 has been chosen and GIZA++ 1.0.3 for the word alignment. Both are state of the art implementations for their respective tasks. However, as each of them represents only one component in the whole pipeline, they can be replaced any time more appropriate tools appear. The implementation of these tools as Web services has been done in a general way so the same methods are applicable to other tools in the same manner and is therefore to be seen as proof of concept.

For the creation of Web services, the Soaplab suite (see section 4.1.1) in its currently most recent version 2.2.0 has been chosen. All the tools applied in this work allow the free use and adaptation of the source code under certain conditions. The GNU General Public License (GPL)[1] (used by Hunalign) requires that changes to the source code are published under the same license. The Apache License[2] (used by Soaplab) instead, does not require re-publish changes to the source code, but does require attribution to the original software.

The choice for Soaplab has been motivated by its ease of creating Web services from existing tools and by its platform independence that comes with the Servlet technology. It has been decided to use Soaplab in the PANACEA project to use it for the same reasons and by many projects in its original field, the bioinformatics, as well. Even though Soaplab is not applicable in a straight forward way for some of the tools used in this work, writing the necessary wrapper scripts is more efficient than completely re-writing these tools as Web services.

[1] GNU General Public License (GPL): http://www.gnu.org/licenses/gpl.html
[2] Apache License: http://www.apache.org/licenses/

```
appl: Square [
  documentation: "Squares a number."
  groups: "Testing"
  executable: "/home/carsten/square.pl"
  nonemboss: "Y"
]

float: number [
  parameter = "Y"
]

outfile: result [
  default: stdout
]
```

Figure 4.1: A simple Soaplab ACD file defining the input parameter *number* and the output *result* for the tool *Square*.

4.1 From Command Line Tools to Web Services

4.1.1 Soaplab

Java servlets (see section 3.4.2) are platform independent as they run on every operating system supporting the Java platform that is available for all commonly used operating systems. However, the technique requires software to be developed as servlet classes in the Java programming language. In the time-frame of this work, it would be not feasible to re-implement tools as Java servlets; neither would this procedure qualify as a good solution in general because the tools applied in this work are already freely available and re-implementing them would not necessarily provide any advance in quality while being prone to the introduction of programming mistakes.

A way to make existing command line tools work as Web services without the need for re-implementing them 'is to write a wrapper around them, making the command line tools unified, remotely accessible, and hiding their dependencies on the underlying operating system' (Senger, Rice, and Oinn, 2003).

In order to create a Web service from an existing command line tool, Soaplab provides the tool *acd2xml*. It reads a file in the ACD format that defines the command line tool's basic parameters such as the supported inputs and outputs (see figure 4.1). The ACD format has been specified in the context of the EMBOSS project (*The European Molecular Biology Open Software Suite*) (Rice et al., 2000). The example shown in figure 4.1 links to the simple Perl script *square.pl* that reads a number as command line argument and returns its square value (see figure 4.2).

The section starting with `appl` in figure 4.1 defines the service's basic information. The parameters `documentation` and `groups` are used to provide a brief description and to assign the service to a group of applications respectively. The attribute `executable` links to the actual command line tool callable in the local file system. The boolean expression `nonemboss: "Y"`

```
#!/usr/bin/perl

my $input = $ARGV[0];
my $result = $input * $input;
print "$result\n";
```

Figure 4.2: The Perl script *square.pl* that returns the square of a given number.

declares that this programme is not part of the EMBOSS software suite, implying that it does not natively use the EMBOSS internal formats.

The following section, `float: number`, declares the input variable named *number* and defines its data type. Apart from *float*, primitive standard types such as *string*, *boolean*, and *integer* are available, as well as lists. Additionally, the data type *infile* allows for an input file to be uploaded to the Web service. The *filelist* type does the same for multiple files. The input files can be defined with either `comment: "data direct"` or with `comment: "data filename"`. In the former case, files will be uploaded directly by the user while in the latter case, he will only specify a file name.

Each input parameter for a command line tool is declared in the same syntax as `number` in figure 4.1: the data type followed by the variable name. By default, Soaplab uses the variable name to specify the parameter in the form `-<variable name> <value>`, e.g. `square.pl -number 2`, when calling the respective tool. To avoid the appearance of the variable name on the command line, the ACD attribute `parameter: "Y"` can be set for an input variable; this also makes the variable mandatory. When using the attribute `standard: "Y"`, an argument is defined as mandatory as well, but appearing with the variable name as command line switch in the default manner. `addition: "Y"`, however, declares an argument to be optional.

The parameter `qualifier` allows to specify a command line switch to be used instead of the variable name. This is helpful when a switch name is not meaningful (e.g. `-f`), making it hard for the user to figure out its purpose (e.g. the specification of an input file). In that example, `file` could be chosen as the input name while `qualifier: f` would specify the switch to be used when calling the command line tool.

If the pre-defined command line parameter formats do not match a tool's requirements, they can be customised with the attributes `tagsepar` and `template` using the strings `$$` and `&&` as placeholders for the parameter value and name respectively. Additionally, an input value can be hidden from the Web service user but still be passed on the command line with the attribute `comment: "display false"`.

Eventually, the output is configured in the section `outfile: result` (see figure 4.1). This corresponds to the name under which this output is made available by the Web service. In the example from figure 4.1, the result is returned to the Unix output stream. Messages written to standard error can be returned as well or can be investigated in the file *report.txt* that is created automatically. The optional parameter `comment: bindata` allows to declare that binary encoded data will be returned. In that case, results are printed into a file and its name can be

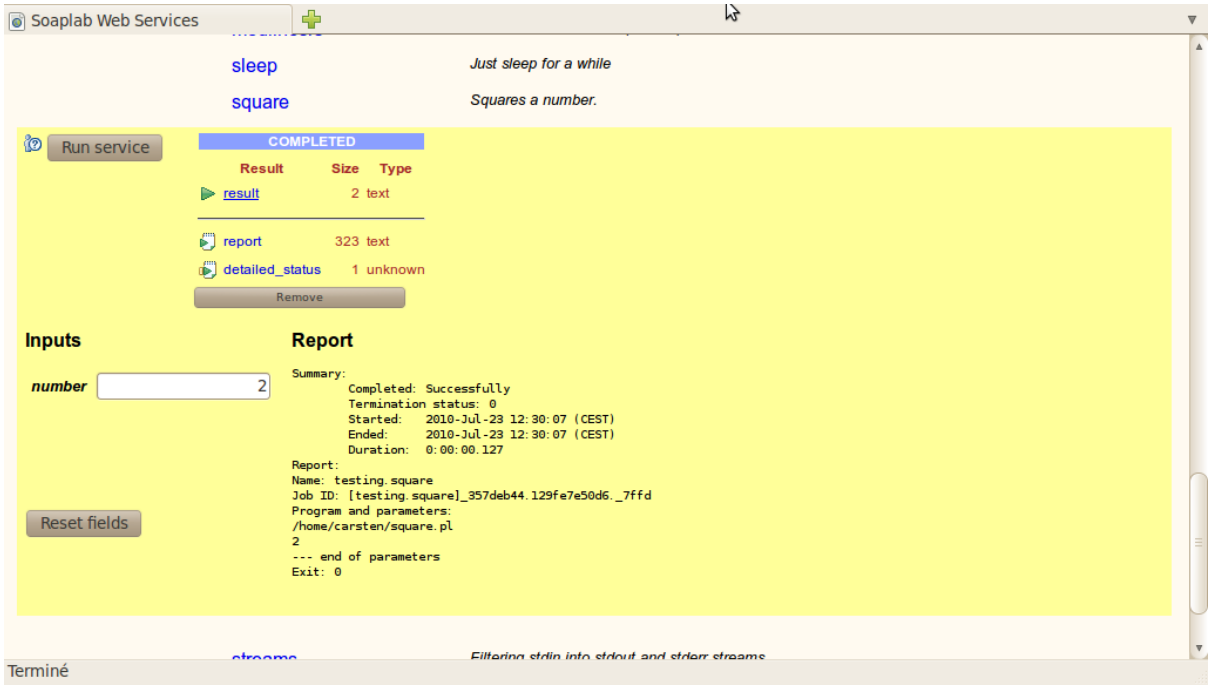

Figure 4.3: The *square.pl* script as a Web service provided by Soaplab on a Tomcat server.

specified in a separate `outfile` section to create an additional output channel.

The Soaplab tool *acd2xml* creates XML files in the Soaplab metadata format from any ACD file. Soaplab uses this data to build a Java servlet file `soaplab2.war` that can then be deployed to and provided by a servlet container. Figure 4.3 shows the newly created service *square* accessed on an Apache Tomcat server with the web-based client Spinet that is part of the Soaplab suite. The users enters the input into the text field labelled *number* and the result is stored in the file *result* that is accessible over a link from the web interface.

Soaplab automates the integration of all services defined in ACD files using Apache Ant[3], a Java-based tool for the automatic compilation of software projects in various languages. The command `ant gen` recompiles the Soaplab servlet including all services found in the subdirectory `src/etc/acd`. The result file can then be deployed to the appropriate Tomcat directory using `ant jaxdeploy`. As an alternative to the Jax implementation of SOAP, the Axis protocol can be used with `ant axis1deploy`.

4.1.2 Integrating Components

This section describes the implementation of Web services using Soaplab. For each service, an ACD file is written. Where necessary, wrappers are implemented in addition. Figure 4.4 illustrates an example pipeline that includes the most relevant alignment components.

[3]Java Ant: http://ant.apache.org/

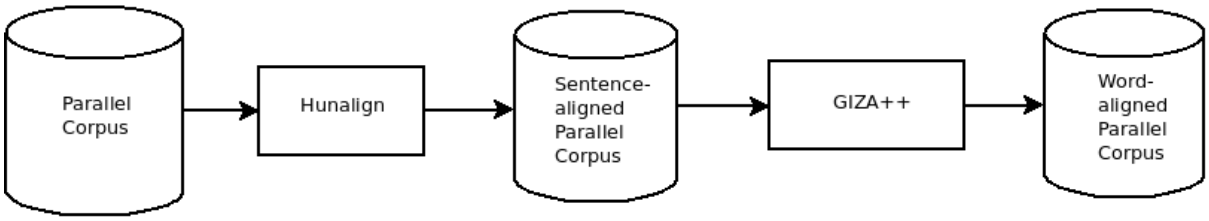

Figure 4.4: The Pipeline Leading from a Parallel Corpus to a Word-aligned Corpus.

Hunalign option	Description
-text	Print output in text format
-bisent	Print 1:1 alignments (bisentences) only
-cautious	Only print alignments that are followed and preceded by 1:1 alignments
-hand=file	Compute precision and recall for the alignment using *file* as a gold standard
-realign	Apply iterative algorithm
-autodict=filename	Save dictionary built during alignment to *filename*
-utf	Use UTF-8 encoding for the input files
Postprocessing Options	
-thresh=n	Don't print alignments with score lower than $n/100$
-ppthresh=n	Don't print rungs with an average score of less than $n/100$ in their vicinity
-headerthresh=n	Filter rungs with score of less then $n/100$ at the end and beginning of texts
-topothresh=n	Filter rungs with less than $n\%$ 1:1 alignments in their vicinity

Table 4.1: The non-mandatory options for Hunalign (source: Hunalign manual).

Hunalign

Hunalign has three mandatory arguments: one dictionary file and two files containing the parallel texts. In a Web service, all of these ought to be realised as files that are uploaded by the user. In the ACD format, this corresponds to the data type `infile`; in order to require the user to upload the files rather than providing a reference, the parameter `comment: "data direct"` is set for these input variables. Additionally, the attribute `parameter: "Y"` is set for all of the input parameters to declare them as mandatory.

Apart from the mandatory ones, Hunalign accepts numerous optional parameters (see table 4.1); in ACD, they are set with the attribute `addition: Y`. Some of them are boolean (`-text`, `-bisent`, `-realign`, and `-utf`) and therefore do not require any values. Their presence or absence suffices to manipulate the programme behaviour. In the ACD format, input arguments of this kind are reflected by the data type *boolean*.

The optional `-hand` parameter allows the specification of a gold standard alignment in an additional input file. If provided, Hunalign prints an evaluation of the result where its output is compared to the given gold standard.

Numerous threshold parameters are specified as integer input values. These postprocessing

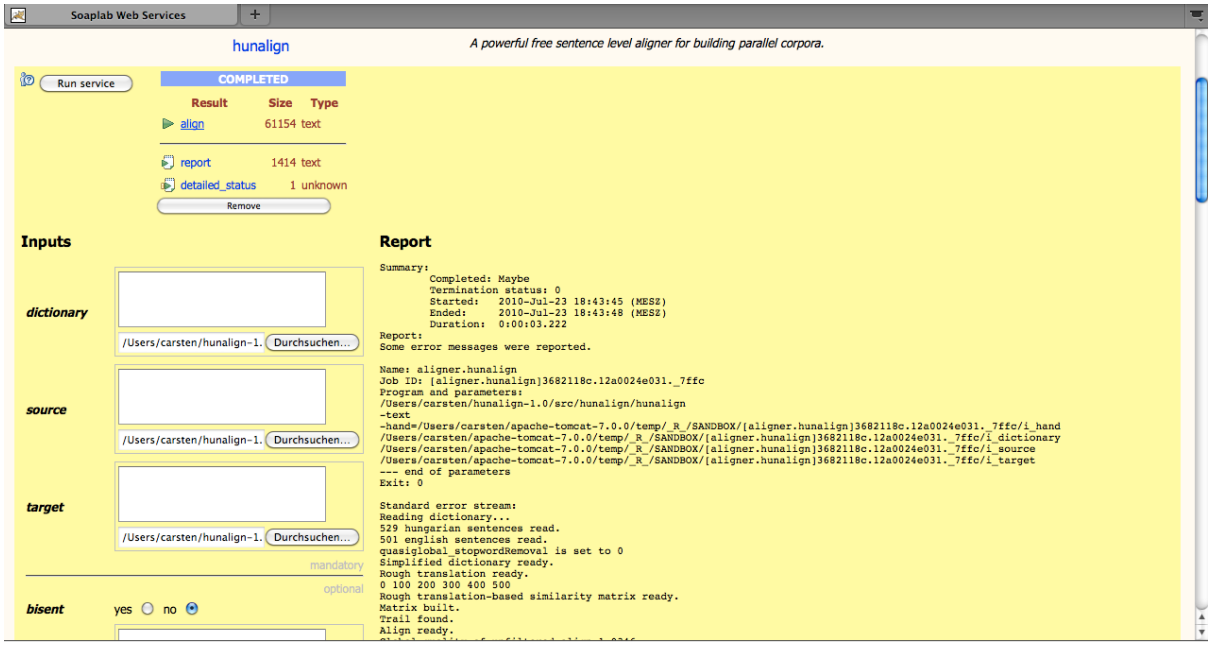

Figure 4.5: The Hunalign Web service called with the web-based Spinet client.

filters become relevant after the alignment process by removing results with low score (`-thresh`), rungs with low average score in their vicinity (`-ppthresh`), bad rungs at the beginning and the end of texts (`-headerthresh`) and rungs with too few 1:1 alignments in their vicinity (`-topothresh`).

An ACD file for Hunalign including all command line parameters has been implemented (see appendix A.2). Figure 4.5 shows the alignment Web service being called with the Spinet web client from the Soaplab suite (see section 4.2.1).

Hun2Giza

The Perl script *hun2giza.pl* written during this work (see appendix A.3) converts a parallel text produced by Hunalign into two files that are usable as input for GIZA++. In the Hunalign output file, each line comprises three elements, separated by tabs (see figure 4.6). The first two columns contain the source and the target language sentences. The words can be represented by numbers or, if Hunalign has been executed with the optional parameter `-text`, the full sentences are provided. The third column contains a confidence estimation value for the given alignment; it is omitted during the conversion because GIZA++ does not make use of this information.

```
szomoru szurke nap volt ez a nyugtalansag elegedetlenseg          he be a sad
    day a day of grey unrest of discontent          1.52083
```

Figure 4.6: A parallel Hungarian-English sentence pair as output by Hunalign in text mode (Hungarian diacritics removed).

Hun2Giza reads a Hunalign output file and prints the sentences of one of the languages to

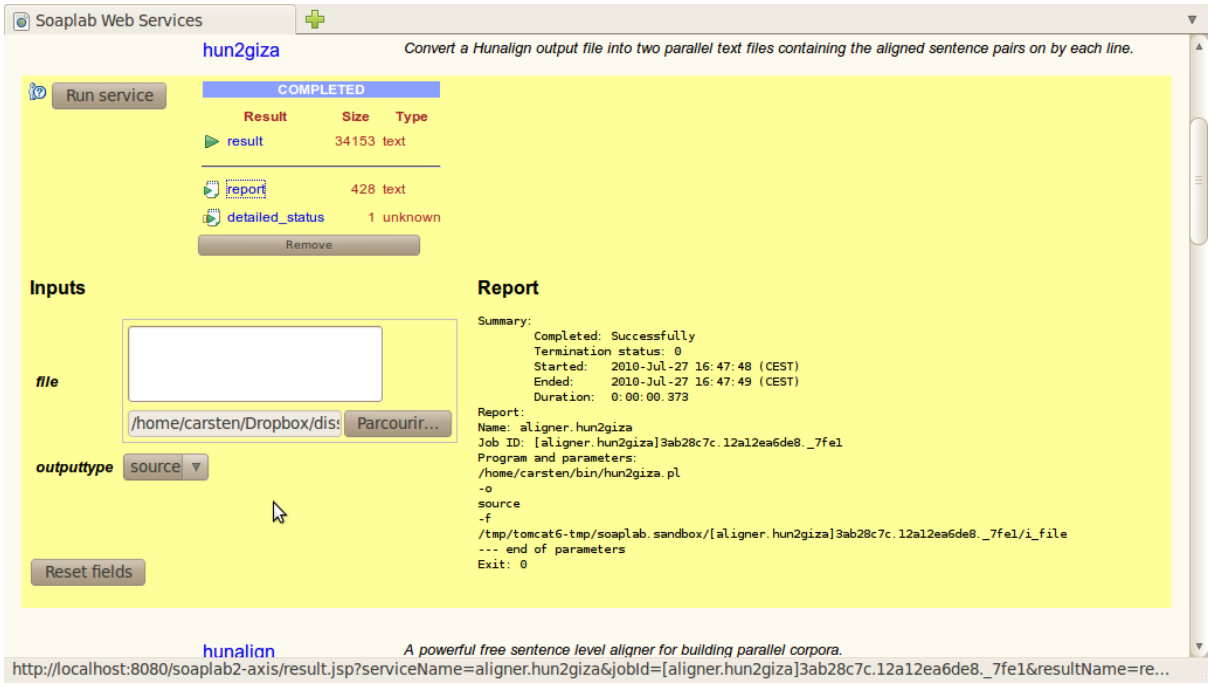

Figure 4.7: The Hun2Giza Web service called by the web-based Spinet client.

the standard output. Both languages can be output separately in order to allow storing them in different files that provide aligned sentences that can be used by GIZA++. Every line in the output files contains one sentence that corresponds to the sentence in the other file that is stored in the same line number. In the case of 1-to-n alignments where $n > 1$, the first line of the entry contains the actual sentences while the following $n - 1$ lines for the same alignment remain empty.

With the command line switch `-o [source|target]`, the user specifies the sentences of which language he wants to retrieve. Commonly, both will be required so the tool would be called twice; once with each of the output option. The input file is specified with the `-f` switch.

The ACD file for creating a Soaplab-based Web service from Hun2Giza is shown in appendix A.4. It defines two input ports corresponding to the parameters `-o` and `-f` as well as an output port which will contain the target or source sentences, as specified by the output type switch. Figure 4.7 shows the access to the Web service provided by a Tomcat server.

Hun2Giza2

Instead of printing to the standard output channel, Soaplab Web services can also return files written by a software tool. To create a converter that is more practical for the given task, the initial version of *hun2giza.pl* has been changed. The new version (*Hun2Giza2*) prints both aligned target and source sentences directly into files (see appendix A.5). The output file names can be specified by the user with the second and third command line parameters – the first one is the input file – or will be defaulted to `source` and `target`.

This change also requires a new ACD file that reflects the tool's new output channels (see

appendix A.6). The sections `outfile: source` and `outfile: target` refer to the files containing the sentences in the different languages. The servlet container will store them in its local file system when executing the tool and make them accessible to the user afterwards.

plain2snt

The tool *plain2snt.out* is part of the GIZA++ package. It takes two input files as arguments, expecting that the first one contains sentences in the source language and the second one contains translations of these sentences in the target language. Corresponding sentences are assumed to appear in the same line in both files.

The tool writes four output files: a vocabulary file for both of the input files and two GIZA++ sentence files. The sentence files contain the aligned sentences in a numeric encoding in both directions (source-target and target-source) respectively.

The names *plain2snt.out* assigns to the output files is subject to the input file names. This makes it impossible to return them automatically because the ACD format cannot deal with variables that could, for instance, derive the output file names from the input file names. Therefore, the wrapper script *plain2snt.sh* has been implemented that calls *plain2snt.out* but allows for the explicit specification of the output file names on the command line. *plain2snt.sh* predicts the output file names generated by *plain2snt.out* and uses the Unix tool *cat* to print their content into the files specified by the user (see appendix A.7).

plain2snt.out generates the output file names according to a simple pattern: it adds the extensions `.vcb` and `.snt` to the input file names; if they end with `.txt` or `.tok`, these extensions are removed before attaching the suffix. The wrapper script's main task is to detect suffixes `.txt` and `.tok` from the input files as they change the name generation behaviour of *plain2snt.out*. Subsequently, the script adapts the predicted output file names accordingly.

In the corresponding ACD file (see appendix A.8), two input ports are defined for the first two command line arguments referring to the input files. Four output ports are created to grant access to all the files created by *plain2snt.out*.

mkcls

The tool *mkcls* (Och, 1995) is a part of the GIZA++ package, but it is independent from the aligner. *mkcls* automatically extracts word classes from a text corpus using a maximum likelihood criterion. The user can specify the number of classes to generate (switch `-c`) and the number of iterations the algorithm will perform (`-n`). The input file is any text – usually the text to be aligned with GIZA++ in the subsequent step – and it is specified with the command line switch `-p`. An output file must be specified with `-V` or the results will not be stored.

Appendix A.9 shows the ACD file wrapping *mkcls* as a Soaplab Web service. It specifies the two optional parameters and the mandatory input file as well as the output file containing the classes. Word class files for both the source and the target language are not necessary for the GIZA++ alignment process, but they will improve the results if available. It is therefore recommended to run *mkcls* on both of the initial input text files.

GIZA++

GIZA++ is completely configurable over a configuration file; its location can be specified on the command line. All of the 91 options (GIZA++ version 1.0.3) can be defined there, but may also be overwritten on the command line when calling GIZA++. The configuration file comprises one option per line in which the first column contains the option and the second column – separated by a space – provides the corresponding value. In order to overwrite a value, the option name serves as a command line switch which is followed by the new value, e.g. `GIZA++ -p 0.98`.

An example entry for the output alignments is shown in figure 4.8: the first line provides information about the sentence lengths and the alignment score, the second line shows the target sentence and the third line presents the source sentence, each word being annotated with none, one, or several numbers that refer to word positions in the target sentence. Note that the sentences in this example have been preprocessed by Hunalign's built-in stemmer which yields the example sentences presented in figure 4.8.

```
# Sentence pair (7) source length 8 target length 12 alignment score :
    4.53448e-17
he be a sad day a day of grey unrest of discontent
NULL ({ }) szomoru ({ 1 }) szurke ({ 9 }) nap ({ 2 }) volt ({ 6 }) ez ({ })
     a ({ }) nyugtalansag ({ 3 4 5 7 8 10 11 }) elegedetlenseg ({ 12 })
```

Figure 4.8: A parallel Hungarian-English sentence pair, word-aligned by GIZA++ (Hungarian diacritics deleted).

In addition to the actual alignment, information such as intermediate results, probability tables, fertility tables, etc., that are stored in other files during the alignment process, contain information that is valuable for some use cases. In order to avoid the loss of this information, the Web service must allow the user to access these files too.

The configuration file can specify the location of the input files, i.e. the vocabulary files for both languages and the file that contains the actual sentences. However, in the case of a Web service, these input files are generally not available in the server's local file system and therefore cannot be specified in the configuration file. Instead, the user needs to upload them and their locations have to be specified when calling GIZA++.

GIZA++ produces numerous files, their common name prefix is defined with the GIZA++ `-o` switch. However, this raises new problems when creating a Soaplab Web service with an ACD file for two reasons:

1. The user does not directly specify an output file name but only the pattern that will be used for all the files created by GIZA++ during the alignment process. As mentioned above, it is not possible to deal with this kind of implicit information in the ACD format, so the output file name cannot be derived from the prefix specified by the `-o` switch.

2. The number and names of produced files partly depends on the configuration options.

The aligned sentences are normally written into the file `<prefix>.A3.final`, but that name changes when running on a case sensitive operating file system (such as HFS+ that is used by Mac OS X) or if that file already exists.

To create an interface with which the ACD format can deal, the wrapper script *giza.sh* has been implemented (see appendix A.10). It takes several command line parameters:

- The source text vocabulary file
- The target text vocabulary file
- The files containing the parallel sentence pairs in GIZA++ format
- GIZA++ configuration file (optional, must end with `.zip`, defaults to `giza.zip`).
- A dictionary in GIZA++ format (two columns separated by spaces defining target word number and source word number) (optional)

The wrapper script promotes the parameters that define the input files to GIZA++ using the appropriate command line switches. The output file name given to *giza.sh* is used to derive an output prefix, e.g. `giza.zip` yields the prefix `giza` that is then included into the GIZA++ command line as `-o giza`. Eventually, the wrapper script collects all the files in the local directory that start with the given prefix and adds them to a Zip archive. The Web service returns this archive to the user to provide him with all available GIZA++ output files. Accordingly, the corresponding ACD file defines five input ports from which one is optional (the dictionary) and one output file (see appendix A.11).

However, the word classes that can be created with *mkcls* are not exploited by the described method to call GIZA++. This is due to the fact that their location is not configurable but statically defined in the GIZA++ source code. The programme searches for them in the current directory by attaching the suffix `.classes` to the provided vocabulary files. These file names are not necessarily preserved in the Web service's local environment even if they were uploaded with suitable names. A solution without having to change the GIZA++ source code is presented in the following subsection.

Wrapping all the GIZA++ Components

The preceding parts of this section have described the tools that prepare the files output by Hunalign for the GIZA++ word alignment process and the GIZA++ execution itself. As shown, this causes problems regarding input and output files because GIZA++ is designed to run on a system with reliable output file names while the file names that are used internally by a Web service are not predictable.

The last two steps, creating word classes and calling GIZA++, are automatically performed by the C-Shell script *trainGIZA++.sh* that is part of the GIZA++ package. However, the script as provided by GIZA++ version 1.0.3 actually does not fully comply with the C-Shell syntax;

```
#!/bin/bash

#PATH containing GIZA++ tools:
GIZAPATH=/home/carsten/bin

# SOURCE and TARGET must not contain . suffices or trainGIZA++.sh won't
    work!
SOURCE=$1
TARGET=$2

# call GIZA tool plain2snt.out, will create vcb and snt files
$GIZAPATH/plain2snt.out $SOURCE $TARGET > plain2snt.out

VCBFILE1=$(basename $SOURCE).vcb
VCBFILE2=$(basename $TARGET).vcb
SNTFILE=$(basename $SOURCE\_$(basename $TARGET).snt)
export PATH=$PATH:$GIZAPATH
$GIZAPATH/trainGIZA++.sh $VCBFILE1 $VCBFILE2 $SNTFILE

cat GIZA++.A3.final
```

Figure 4.9: The script *giza_complete.sh* wrapping around the different GIZA++ tools.

therefore a correction in the line reading the command line arguments hat to be done (change `$#` to `$#argv` in line 3).

A wrapper script *giza_complete.sh* around the tools *plain2snt.out* and *trainGIZA++.sh* has been written that takes the parallel texts produced by Hun2Giza as command line parameters and uses the command `cat` to print the resulting alignment file named *GIZA++.A3.final* to the Unix standard output (see figure 4.9). This allows to run the complete alignment procedure on the server in a single pipeline, including the word class generator *mkcls*. This method does not allow intermediate interaction with the user; this can be seen as an advantage because it eases the application. On the other hand, the user is not able to change all the configuration options. However, this approach solves the file naming problem with word classes generated by *mkcls* that has been described previously in this section.

In the same time, executing this complete GIZA++ workflow at once forbids the user to inspect intermediate output files or to use additional features such as the usage of an optionally provided word-based dictionary. Another risk this approach takes is that it defines the name of the file it returns statically although it can possibly change depending on the environment in which GIZA++ runs. However, this issue can be resolved by adding a subroutine that automatically searches for the correct file; in the context of this work, the script runs on the same case-sensitive file system and in its own directory, so the output file always gets the same name.

4.2 Accessing and Using Web Services

This section shows how to use the Web services created in the previous sections in practice. Once the Soaplab and its Web services have been deployed to a servlet container as described in section 4.1.1, they are accessible with any client that is capable of communicating via the chosen SOAP implementation, i.e. Jax or Axis.

By default, Apache Tomcat provides the Soaplab service using Jax at the directory `soaplab2` if it has been deployed with the command `ant jaxdeploy`. When using the command `ant axis1deploy` instead, using the Axis protocol, the default location on the server is `soaplab2-axis`.

At first, a client requires information about a Web service's basics such as the communication protocol and the available input and output ports. The typical source for this information is the WSDL file. In the context of Soaplab, the WSDL file is stored at an address that is generated from the service's name. For instance, for the previously created service *hunalign*, that has been defined as part of the group *aligner*, Soaplap provides a WSDL file at this location on the web server:

```
soaplab2-axis/services/aligner.hunalign?wsdl
```

In order to make a complete URL, the protocol, the server name and optionally the port need to be included. If the server is running locally (`localhost`), is listening at port number 8080, and uses the HTTP protocol, the full URL looks like this:

```
http://localhost:8080/soaplab2-axis/services/aligner.hunalign?wsdl
```

To access any other service, the service name (`hunalign`) and its group name (`aligner`) need to be adapted. A list of all services that are made available by a Soaplab server is provided at this URL:

```
http://localhost:8080/soaplab2-axis/services
```

4.2.1 Soaplab Built-in Clients

Spinet

Soaplab provides a simple web-based client to allow instant access to its Web services: Spinet. In practice, this mainly serves testing and demonstration purposes as Web services are usually designed for automatic access by other software programmes. In Spinet, the user uploads input parameters for a Web service via HTML forms; they can handle text boxes, selection lists, file uploads, etc.

After a Web service has run, Spinet makes the results accessible through HTML links pointing to the resulting file or files. Error and other diagnostic messages are accessible on the Spinet page in the *Report* section. Several figures showing Spinet accessing Soaplab Web services have been shown in section 4.1.

Soaplab Command Line Client

In addition to the web-based client, the Soaplab suite provides a command line client (*run-cmdline-client*) for its Web services. Server and protocol are specified with the command line switches `-host`, `-port`, `-protocol` (can be `jaxws`, `local`, or `axis1`), and `-name` (e.g. `aligner.hunalign`). Alternatively, the full URL can be specified with `-e`.

The standard use case, executing a job and waiting for it to end, is invoked with the `-w` switch. All results will be printed to the screen if `-r` is specified as well. Alternatively, `-i` returns information about the input parameters.

Input parameters are specified at the end of the command line as pairs of parameter names and values. If the content of a file should be uploaded, the file's name preceded by '`:`' can be used as input value. An example call for the `square` Web service shown above with `2` as input value:

```
run-cmdline-client -protocol axis1 -host localhost -port 8080 \
    -name testing.square -w -r number 2
```

4.2.2 Taverna

> The Taverna project provides a graphical workbench tool for both creating and running workflows that represent *in silicio* bioinformatics experiments. (Oinn et al., 2004)

Despite its origins in the field of bioinformatics, Taverna (Oinn et al., 2006) is applicable for any use case that involves Web services. Therefore, the PANACEA project has decided to define Taverna as its default workflow editor, meaning that it will be the standard tool with which the users plug different Web services together to create customised pipelines for their specific needs.

> In Taverna, a workflow is considered to be a graph of processors, each of which transforms a set of data inputs into a set of data outputs. (Oinn et al., 2004)

A processor, in the Taverna terminology, transforms some input to some output data and, in the context of this work, corresponds to a Web service. The processor's name also reflects the Web service's name; in the case of Soaplab Web services, these are the names that have been defined in the ACD definition files. Additionally, Taverna allows other processor types that are not applied in this work such as locally installed programmes and nested workflows. A workflow can have multiple input and output ports that form the beginning and the end of the pipeline while each component has its own input and output ports.

Using Taverna has several advantages for users of NLP Web services. The graphical interface allows it to easily apply numerous software tools and connect their outputs with the inputs of others. As for Web services in general, the user only requires background knowledge about the meanings of the tool's inputs and outputs, but no technical knowledge about the software.

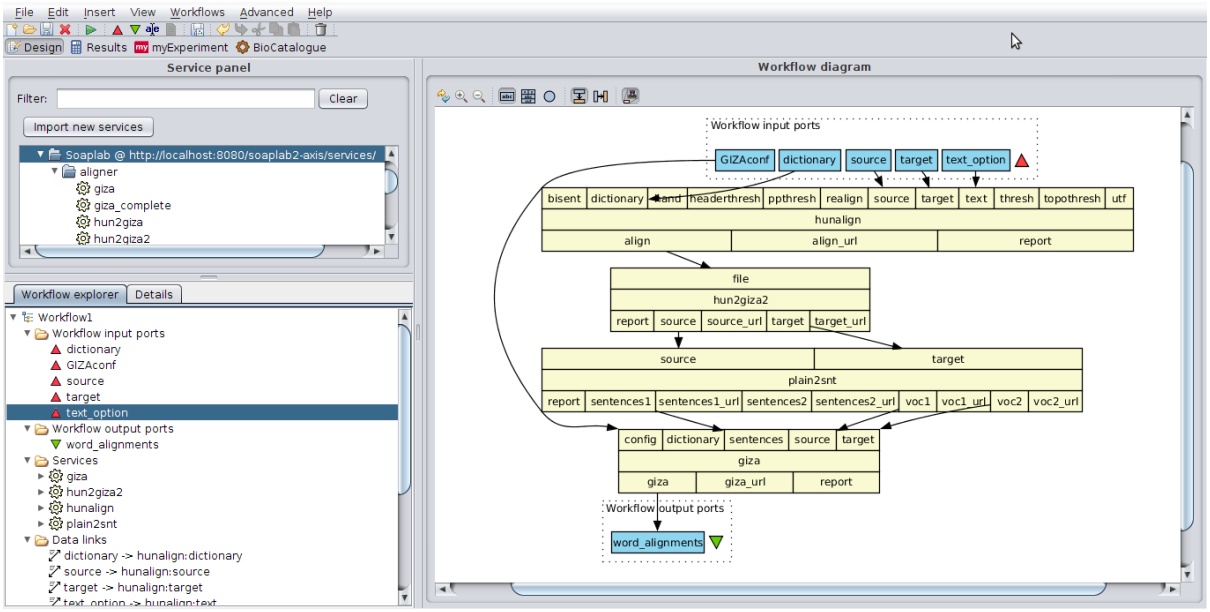

Figure 4.10: A Taverna workflow implementing an alignment pipeline.

However, Taverna allows the user to investigate intermediate results as Taverna temporarily stores them locally and displays them in its graphical user interface upon request.

Workflows created with Taverna are stored in the XML-based markup language SCUFL (*Simple Conceptual Unified Flow Language*). Being a text-based format, these file are human-readable and could be used and edited by third-party editors as well. SCUFL is described in the Taverna manual[4].

Taverna supports WSDL-based Web services in general; they can be added to the Taverna service provider list using the WSDL URL specified by Soaplab as described above. Additionally, Taverna version 2 directly supports Soaplab servers which makes it possible to integrate a Soaplab server with all its services at once. It can be accessed in the sub-directory `services` (by default `soaplab2-axis/services/` when using the Axis protocol). For Taverna version 1, there is a Soaplab plugin available[5].

After adding the Soaplab server's Web services, they can be integrated as components of a Taverna workflow. Figure 4.10 shows a complete workflow using the tools and Web services created during this work. In the example, the input ports stand on the top. The user can specify as many as necessary for his needs; here five have been specified. The two input corpora (`source` and `target`) and a dictionary (`dictionary`) form the basic parameters for Hunalign. Additionally, the input `text` makes Hunalign produce text output rather than numeric. These four workflow inputs are directly connected with matching input ports of the first processor (`hunalign`). For the fifth input port (`GIZAconf`), the GIZA++ configuration file is expected from the user and it is therefore directly connected to `giza`, the last processor in the pipeline.

For each processor, the diagram shows all available input and output ports (this can be

[4]MyGrid: http://www.mygrid.org.uk/usermanual1.7/scufl_language_wb_features.html
[5]Taverna Soaplab plugin: http://soaplab.sourceforge.net/soaplab2/TavernaNotes.html

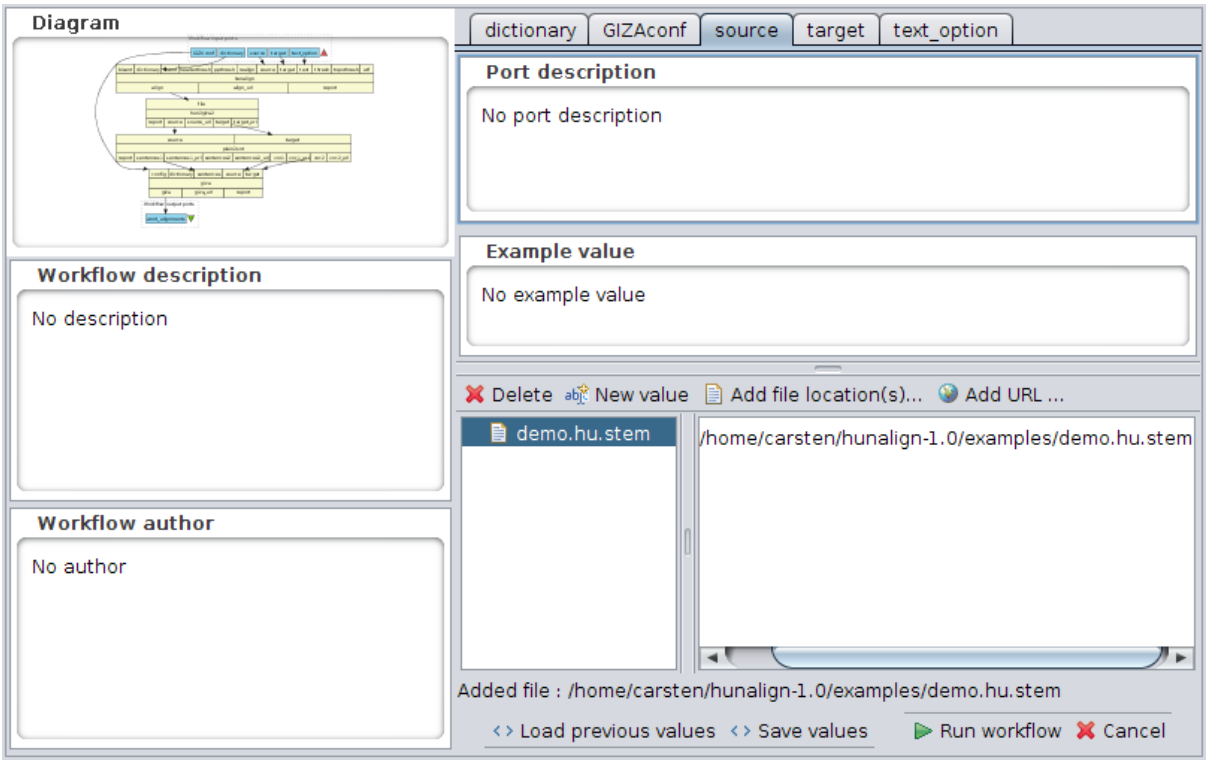

Figure 4.11: The Taverna workflow input is specified by the user when executing the process.

deactivated to make large workflows clearer). The `hunalign` processor's output is connected to the only `hun2giza2` input port. As described in section 4.1.2, Hun2Giza2 produces two files, reflected by the output ports `source` and `target`, that are then forwarded to the matching `plain2snt` input ports. In a similar manner, the three relevant `plan2snt` output ports (two vocabulary files `voc1` and `voc2` and the aligned sentences file `sentences1`) are connected to the corresponding input ports of the `giza` component. Finally, the GIZA++ output is returned to the user via the connection between the `giza` output also named `giza` and the workflow output port `word_alignments`.

Taverna makes the final result(s) available in its interface after the workflow has terminated where they can directly be investigated or stored. The intermediate results produced by the single components are available too.

The input values are specified by the user when she runs the workflow (see Figure 4.11). Note that the data types are not validated by Taverna so it falls into the user's responsibility to provide the data in the form required by the services. However, for the connection between input and output ports, Taverna can provide a validation report in which conflicting data types are reported.

4.2.3 Programmatic Access

The purpose of most Web services is to provide functionality that is directly usable by other software programmes. These other programmes therefore need to send a request to the Web

```
#!/usr/bin/perl

use SOAP::Lite;

my %input = ('number' => $ARGV[0]);
my $uri = 'http://localhost:8080/soaplab2-axis/services/testing.square';
my $soap = new SOAP::Lite->proxy($uri);
my $jobid = $soap->createAndRun(SOAP::Data->type(map=>\%input))->result();
my $results = $soap->getResults($jobid)->result();
print "Square of $number: $$results{'result'}";
```

Figure 4.12: A Web service client that accesses the *square* Web service using the Perl module SOAP lite.

service provider according to its capabilities, i.e. using SOAP in the context of this work. The developer may implement the SOAP communication from scratch, entirely rely on existing libraries or remain between these two extremes.

For demonstration purposes I have implemented a very simple client programme in Perl that accesses the previously presented *square* Web service (see figure 4.12). It uses the Perl module Soap Lite[6] that provides most functionality that is necessary for using SOAP Web services. The programme simply reads a number from the command line, sends it to the statically specified server, retrieves the results, and prints it to the screen.

The example initially creates a SOAP client object with the Soap Lite `proxy()` method that takes the Web service's URL as an argument; in this case it is running locally, i.e. on `localhost`. Afterwards, the service's function `createAndRun()` is executed with the input parameter `number` as argument that is read from the command line and stored in the associative array `\input`; the function `result()` is called immediately in the same line and returns the numeric id of the started job that is used to control the job. It allows to query the job's status – is it completed or still running? – and to react accordingly.

The code shown in Figure 4.12 requests the result with the function `getResults($jobid)` and prints it to the screen. It returns a reference to an associative array that contains, besides `result`, the keys `report`, `detailed_status`, and `result_url`.

4.3 Concluding Remarks

It has been shown in this chapter how any command line tool can be implemented as a Servlet-based Web service. Soaplab makes this task easy in most standard cases, although the underlying ACD format is not sufficient for output files that are not definable by the user. Examples of how to solve this with small scripts wrapping the actual tools have been presented and are generalisable to any other tool.

In summary, a fully working pipeline from a parallel corpus via sentence alignment to a corpus aligned on a word level has been developed with state of the art tools. This pipeline

[6]Soap Lite Perl module: http://www.soaplite.com

can be extended by adding other existing tools and its components can be integrated by other developers as well as be replaced by other tools and services that might be available in the future.

Chapter 5

Results and Discussion

5.1 Evaluation

As stated in the introduction to this work, its aim is not to improve the results by the single components involved in the platform, but rather to make related tasks more efficient by improving usability and availability. Therefore, the existing programmes that have been implemented as Web services, i.e. Hunalign and the GIZA++ tools, do not need to be evaluated again; they are well established and have not been changed during this work.

A formal evaluation of the whole platform is a very complex task. There are many unpredictable factors that depend on the user's aims and skills. An intuitive approach to an extrinsic evaluation would be to give a task like 'Create a word-aligned corpus from a parallel corpus using Hunalign and GIZA++' to a number of human users with different background knowledge and provide one group with the command line tools and make the other group solve the task using this platform.

However, Web services are not directly accessible by human users, so they require another tool like Taverna (see section 4.2.2) to create a workflow. Then, objective measures such as the time used for the task and subjective measures such as user satisfaction would produce comparable results. However, this evaluation would mainly allow conclusions about the actual user interface, i.e. Taverna versus command line, but not necessarily correlate to the platform's usefulness.

Evaluation has to come closer to the scenario Web services are generally designed for: being included into third-party software programmes. Providing functionality in a customisable and efficient way is the platform's actual task. In this sense, however, a Web service cannot be compared to a command line tool as it is rather an additional functionality for specific needs; a user who needs to run Hunalign and GIZA++ only once will not use a Web service and design a workflow.

Reckoning that a full extrinsic evaluation of the platform is impossible in the context of this work, the objectives that have been defined in section 1.2 will be evaluated in the following sections.

5.1.1 Modularity

All the components in this platform are independent of each other. This implies that they can be used in any order and number necessary. However, new modules can be provided but there is no formal validation, i.e. the developer has to take care of compatibility of the input and output interfaces. The modularity criterion has therefore been fulfilled with the limitation that the syntactic compatibility has to be guaranteed manually.

5.1.2 Usability

To formally test Web services for *usability* and *semantic compatibility*, a model workflow is generated in (Martens, 2003). A workflow module (here: a Web service) M is called *usable*, 'iff there exists at least one environment U that utilizes M' (Martens, 2003). This has been shown for the platform developed here by using all implemented Web services in the example pipeline that makes use of all the components.

The second criterion, *semantic compatibility* is defined in relation to a Web service's environment: if a system out of two components is *usable*, these components are semantically compatible. This has been shown in the example pipeline as well, as the involved components are able to work together in a workflow and to exchange data.

5.1.3 Robustness

Large data

The platform has been tested with the Hungarian-English test data provided by the Hunalign package (30 kilobytes) and the French-English part of Europarl version 5 (315/278 megabytes respectively) and works with both. This has been tested in a local network only. Additional problems might arise when transferring data via a potentially slow and disrupted Internet connection. However, this is a network technology issue and could not be tested systematically here because this lies beyond this work's focus.

Heavy load

An example programme that accesses the alignment Web service sequentially has been executed five times in parallel which led to higher response times, but no substantial problems for the server. Further tests have not been made as this issue depends mainly on the web server and servlet container (in this work: Tomcat) rather than on the platform and its Web services.

5.1.4 Generalisability

This work has shown how to use Soaplab and the ACD format for any command line tool. In the simplest case, it is sufficient to describe the tool's inputs and outputs in a corresponding ACD file. In more complex cases, it is necessary to write a wrapper around the tool that makes the output file names definable by the user. It has been shown how to write wrapper scripts

for the complex case too. These methods can be applied for any command line tool where it is somehow possible to detect the output file names.

5.1.5 Platform Independence

The choice for the Java-based servlet technology has made the platform available for any operating system for which the Java platform and a servlet container exist. Servlets are deployable to any servlet container by design. It has not been tested systematically in this work whether the generated Soaplab servlet is actually usable on every platform, but this can be expected according to the specification of servlets.

5.1.6 Integration

For the central components of the pipeline, Hunalign and GIZA++, the output of the first is not directly usable as input for the second. In order to solve the problem, the format conversion tools Hun2Giza and Hun2Giza2 have been developed so that all components can be used in one workflow.

5.2 Discussion and Future Work

It has been shown how to develop a collection of Web services to make existing language processing tools available on the web and more easily usable. They can be applied in any order and selection (modularity) and the methods can be applied to any command line tool, not only in the field of NLP (generalisability). Additional tools to validate the interface compatibility of the involved tools have been developed (integration). The platform can deal with large amounts of data and serve multiple users at once (robustness) but does not deal with potential networking problems and limitations in computational resources.

This leads to the conclusion that robustness is a potential problem when practically dealing with Web services and large data. Slow connections can make the upload of large corpora a time-consuming bottleneck; in the worst case, network interruptions make it even impossible to use a Web service. To improve robustness, techniques need to be developed that deal with networking problems.

An obvious starting point would be to exchange compressed files that are then extracted locally to reduce the transfer sizes. Furthermore, an idea would be that users can provide their data on some statically available web site and only provide the link; the service would try to fetch the resources repeatedly if necessary. This does not reduce transfer time but – if the user has such a static web site to his availability – would increase usability because the platform took care of the re-initiation of interrupted transfers. However, this issue roots in the underlying networking technology and, again, lies outside the focus of this work.

Regarding modularity, it would be desirable to define a format in which all involved services accept and provide data as inputs and outputs respectively. Considering that all the platform's modules deal with corpora only (rather than, for instance, dictionaries, transfer grammars,

etc.), XCES (Ide, Bonhomme, and Romary, 2000) could be applied for this purpose. It is well established and provides annotation tags for all the information typically used in a corpus. A validation component would then be able to check a new module's output for compatibility with this format and certificate compatibility. However, XCES does not support the marking of comparable corpora that can also form a kind of data to be used in a future processing pipeline.

Appendix A

Appendix: Files

A.1 Extract from the WSDL File for the Hunalign Web Service

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost:8080/soaplab2-axis/
    services/aligner.hunalign" xmlns:apachesoap="http://xml.apache.org/xml-
    soap" xmlns:impl="http://localhost:8080/soaplab2-axis/services/aligner.
    hunalign" xmlns:intf="http://localhost:8080/soaplab2-axis/services/
    aligner.hunalign" xmlns:soapenc="http://schemas.xmlsoap.org/soap/
    encoding/" xmlns:tns1="http://share.soaplab.org" xmlns:wsdl="http://
    schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/
    wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
 <wsdl:types>
  <schema targetNamespace="http://share.soaplab.org" xmlns="http://www.w3.
      org/2001/XMLSchema">
   <import namespace="http://localhost:8080/soaplab2-axis/services/aligner.
       hunalign"/>
[...]
  </schema>
 </wsdl:types>
   <wsdl:message name="describeResponse">
      <wsdl:part name="describeReturn" type="xsd:string"/>
   </wsdl:message>
[...]
   <wsdl:portType name="AnalysisService">
      <wsdl:operation name="run" parameterOrder="jobId">
         <wsdl:input message="impl:runRequest" name="runRequest"/>
         <wsdl:output message="impl:runResponse" name="runResponse"/>
         <wsdl:fault message="impl:SoaplabException" name="SoaplabException
             "/>
      </wsdl:operation>
```

```
  </wsdl:portType>
[...]
  <wsdl:binding name="aligner.hunalignSoapBinding" type="
      impl:AnalysisService">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/
        soap/http" />
    <wsdl:operation name="run">
      <wsdlsoap:operation soapAction="" />
      <wsdl:input name="runRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
            encoding/" namespace="http://axis1.protocol.services.soaplab
            .org" use="encoded" />
      </wsdl:input>
      <wsdl:output name="runResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
            encoding/" namespace="http://localhost:8080/soaplab2-axis/
            services/aligner.hunalign" use="encoded" />
      </wsdl:output>
      <wsdl:fault name="SoaplabException">
        <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/
            encoding/" name="SoaplabException" namespace="http://
            localhost:8080/soaplab2-axis/services/aligner.hunalign" use=
            "encoded" />
      </wsdl:fault>
    </wsdl:operation>
[...]
    <wsdl:operation name="createAndRun">
      <wsdlsoap:operation soapAction="" />
      <wsdl:input name="createAndRunRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
            encoding/" namespace="http://axis1.protocol.services.soaplab
            .org" use="encoded" />
      </wsdl:input>
      <wsdl:output name="createAndRunResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
            encoding/" namespace="http://localhost:8080/soaplab2-axis/
            services/aligner.hunalign" use="encoded" />
      </wsdl:output>
      <wsdl:fault name="SoaplabException">
        <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/
            encoding/" name="SoaplabException" namespace="http://
            localhost:8080/soaplab2-axis/services/aligner.hunalign" use=
            "encoded" />
      </wsdl:fault>
[...]
    <wsdl:operation name="runAndWaitFor">
```

```
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="runAndWaitForRequest">
               <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
                   encoding/" namespace="http://axis1.protocol.services.soaplab
                   .org" use="encoded"/>
            </wsdl:input>
            <wsdl:output name="runAndWaitForResponse">
               <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
                   encoding/" namespace="http://localhost:8080/soaplab2-axis/
                   services/aligner.hunalign" use="encoded"/>
            </wsdl:output>
            <wsdl:fault name="SoaplabException">
               <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/
                   encoding/" name="SoaplabException" namespace="http://
                   localhost:8080/soaplab2-axis/services/aligner.hunalign" use=
                   "encoded"/>
            </wsdl:fault>
         </wsdl:operation>
[...]
         </wsdl:operation>
         <wsdl:operation name="getResults">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="getResultsRequest">
               <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
                   encoding/" namespace="http://axis1.protocol.services.soaplab
                   .org" use="encoded"/>
            </wsdl:input>
            <wsdl:output name="getResultsResponse">
               <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/
                   encoding/" namespace="http://localhost:8080/soaplab2-axis/
                   services/aligner.hunalign" use="encoded"/>
            </wsdl:output>
            <wsdl:fault name="SoaplabException">
               <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/
                   encoding/" name="SoaplabException" namespace="http://
                   localhost:8080/soaplab2-axis/services/aligner.hunalign" use=
                   "encoded"/>
            </wsdl:fault>
         </wsdl:operation>
[...]
      </wsdl:binding>
      <wsdl:service name="AnalysisServiceService">
         <wsdl:port binding="impl:aligner.hunalignSoapBinding" name="aligner.
             hunalign">
            <wsdlsoap:address location="http://localhost:8080/soaplab2-axis/
                services/aligner.hunalign"/>
```

```
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

A.2 The ACD file for Hunalign

```
appl: Hunalign [
  documentation: "A powerful free sentence level aligner for building
      parallel corpora."
  groups: "Aligner"
  version: "1.0"
  executable: "/home/carsten/hunalign-1.0/src/hunalign/hunalign"
  nonemboss: "Y"
]

boolean: text [
  additional: "Y"
  prompt: "The output should be in text format, rather than the default (
      numeric) ladder format."
]
boolean: bisent [
  additional: "Y"
  prompt: "Only bisentences (one-to-one alignment segments) are printed. In
       non-text mode, their starting rung is printed."
]

infile: hand [
  additional: "Y"
  comment: "data direct"
  template: '-hand="$$"'
  prompt: "Specifies a file containing a manual alignment."
]
boolean: realign [
  additional: "Y"
  prompt: "Iterative alignment process."
  information: "If this option is set, the alignment is built in three
      phases. After an initial alignment, the algorithm heuristically adds
      items to the dictionary based on cooccurrences in the identified
      bisentences. Then it re-runs the alignment process based on this
      larger dictionary. This option is recommended to achieve the highest
      possible alignment quality. It is not set by default because it
      approximately triples the running time while the quality improvement
      it yields are typically small."
]
boolean: utf [
  additional: "Y"
  prompt: "Adapt character count to UTF-8 encoded text."
]
integer: thresh [
  additional: "Y"
```

```
  prompt: "Don't print out segments with score lower than n/100."
  template: '-thresh=$$'
]
integer: ppthresh [
  additional: "Y"
  prompt: "Filter rungs with less than n/100 average score in their
      vicinity."
  template: '-ppthresh=$$'
]
integer: headerthresh [
  additional: "Y"
  prompt: "Filter all rungs at the start and end of texts until finding a
      reliably plausible region."
  template: '-headerthresh=$$'
]
integer: topothresh [
  additional: "Y"
  prompt: "Filter rungs with less than n percent of one-to-one segments in
      their vicinity."
  template: '-topothresh=$$'
]
infile: dictionary [
  parameter: "Y"
  comment: "data direct"
  prompt: "Dictionary file. Can be a zero-byte file."
]
infile: source [
  parameter: "Y"
  comment: "data direct"
  prompt: "Segmented source sentences, one sentence per line."
]
infile: target [
  parameter: "Y"
  comment: "data direct"
  prompt: "Segmented target sentences, one sentence per line."
]

outfile: align [
  default: stdout
]
```

A.3 The Script *hun2giza.pl*

```
#!/usr/bin/perl

use strict;
use Getopt::Std;

if (@ARGV != 4) {
  print STDERR "Usage: hun2giza.pl -o [source|target] -f <hunalign ouput
      file>\n";
  print STDERR "Use -o source to retrieve the source sentences and";
  print STDERR "   -o target to get the target sentences.\n";
  exit;
}
our($opt_o, $opt_f);
getopts('o:f:');
unless ($opt_o eq 'source' or $opt_o eq 'target') {
  print STDERR "Bad parameter for -o. Use 'source' or 'target'!\n";
  exit;
}
my $bisentfile = $opt_f;
my $count=0;
open BISENT, "<" . $bisentfile or die $!;
while (my $line = <BISENT>) {
    $count++;
    if ($line =~ /^([^\t]*)\t([^\t]*)\t/) {
      if ($opt_o eq 'source') { print $1 . "\n"; }
      else { print $2 . "\n"; }
    }
    else {
      print STDERR "Error in input file, skipping line $count:\n";
      print STDERR $line;
    }
}
close(BISENT);
```

A.4 The ACD File for *hun2giza.pl*

```
appl: Hun2Giza [
  documentation: "Convert a Hunalign output file into two parallel text
      files containing the aligned sentence pairs on by each line."
  groups: "Aligner"
  nonemboss: "Y"
  executable: "/home/carsten/bin/hun2giza.pl"
]

list: outputtype [
  qualifier: o
  standard: "Y"
  minimum: 1
  maximum: 1
  values: "source;target"
  prompt: "Output source or target sentences."
]

infile: file [
  qualifier: f
  comment: "data direct"
  standard: "Y"
  prompt: "Hunalign output file in text format."
]

outfile: result [
  default: "stdout"
]
```

A.5 The Script *hun2giza2.pl*

```
#!/usr/bin/perl

use strict;

if (@ARGV < 1) {
  print "Usage:";
  print "hun2giza2.pl <Hunalign output file> [source sentences file] [
      target sentences file]";
  exit;
}

my $bisentfile = $ARGV[0];
my $sourcefile = "source";
my $targetfile = "target";
my $count=0;

if ($#ARGV >= 1) {
  $sourcefile = $ARGV[1];
}
if ($#ARGV >= 2) {
  $targetfile = $ARGV[2];
}

open BISENT, "<" . $bisentfile or die $!;
open SOURCE, ">" . $sourcefile or die $!;
open TARGET, ">" . $targetfile or die $!;

while (my $line = <BISENT>) {
    $count++;
    if ($line =~ /^([^\t]*)\t([^\t]*)\t/) {
      print SOURCE "$1\n";
      print TARGET "$2\n";
    }
    else {
      print STDERR "Error in input file, skipping line $count:\n";
      print STDERR $line;
    }
}
close(BISENT);
close(SOURCE);
close(TARGET);
```

A.6 The ACD File for *hun2giza2.pl*

```
appl: Hun2Giza2 [
  documentation: "Convert a Hunalign output file into two parallel text
      files containing the aligned sentence pairs on by each line."
  groups: "Aligner"
  nonemboss: "Y"
  executable: "/home/carsten/bin/hun2giza2.pl"
]

infile: file [
  comment: "data direct"
  parameter: "Y"
  prompt: "Hunalign output file in text format."
]

outfile: source [
  default: "source"
  parameter: "Y"
]

outfile: target [
  default: "target"
  parameter: "Y"
]
```

A.7 The script *plan2snt.sh*

```
#!/bin/bash

GIZAPATH=/home/carsten/giza-pp/GIZA++-v2

if [ $# -lt 6 ]
then
  echo "Usage:"
  echo "$0 <source sentence file> <target sentence file> <source vocabulary
      output file> <target vocabulary output file> <GIZA format sentences
      output file (source_target)> <GIZA format sentences output file (
      target_source)>"
  exit
fi

# strip path name and suffixes .tok and .txt:
INFILE1=$(basename $(echo $1 | sed -e 's/\.txt$//' -e 's/\.tok$//'))
INFILE2=$(basename $(echo $2 | sed -e 's/\.txt$//' -e 's/\.tok$//'))

$GIZAPATH/plain2snt.out $1 $2

cat $INFILE1.vcb > $3
cat $INFILE2.vcb > $4
cat $INFILE1\_$INFILE2.snt > $5
cat $INFILE2\_$INFILE1.snt > $6
```

A.8 The ACD file for *plain2snt.sh*

```
appl: plain2snt [
  documentation: "Call the GIZA++ tool plain2snt which creates vocabulary
      files for both given texts and a GIZA format sentence file."
  groups: "Aligner"
  nonemboss: "Y"
  executable: "/home/carsten/bin/plain2snt.sh"
]

infile: source [
  prompt: "File containing sentences in source language."
  parameter: "Y"
  comment: "data direct"
]
infile: target [
  prompt: "File containing sentences in target language."
  parameter: "Y"
  comment: "data direct"
]

outfile: voc1 [
  prompt: "Vocabulary file for source sentences."
  parameter: "Y"
]

outfile: voc2 [
  prompt: "Vocabulary file for target sentences."
  parameter: "Y"
]

outfile: sentences1 [
  prompt: "Sentences in GIZA++ format (source -> target)."
  parameter: "Y"
]

outfile: sentences2 [
  prompt: "Sentences in GIZA++ format. (target -> source)"
  parameter: "Y"
]
```

A.9 The ACD file for *mkcls*

```
appl: mkcls [
  documentation: "A tool to train word classes by using the maximum
     likelihood criterion."
  groups: "Aligner"
  nonemboss: "Y"
  executable: "/home/carsten/giza-pp/mkcls-v2/mkcls"
]

integer: classes [
  additional: "Y"
  template: "-c$$"
  prompt: "Number of classes to generate."
]

integer: runs [
  additional: "Y"
  default: 1
  template: "-n$$"
  prompt: "Number of optimization runs (larger => better results)."
]

infile: corpus [
  standard: "Y"
  comment: "data direct"
  template: "-p$$"
  prompt: "Filename of training corpus."
]

outfile: output [
  standard: "Y"
  template: "-V$$"
  prompt: "Output classes."
]
```

A.10 The script *giza.sh*

```
#!/bin/bash

GIZAPATH=/home/carsten/giza-pp/GIZA++-v2
OUT=giza.zip
CONFIG=''

if [ $# -lt 3 ]
then
  echo "Usage:"
  echo "  $0 <source vocabulary file> <target vocabulary file> <GIZA++
      sentences file> [<config file>] [<output file>] [<dictionary file>]"
  exit
elif [ $# -ge 4 ]
then
  CONFIG=$4
fi
if [ $# -ge 5 ]
then
  OUT=$5
fi
if [ $# -ge 6 ]
then
  DICT="-dictionary $6"
fi

GIZAOUT=$(basename $OUT .zip)    # output filename extension must be .zip

$GIZAPATH/GIZA++ $CONFIG -S $1 -T $2 -C $3 $DICT -o $GIZAOUT
zip $OUT $GIZAOUT.*
```

A.11 The ACD file for *giza.sh*

```
appl: Giza [
  documentation: "GIZA++ is a program for aligning words and sequences of
      words in sentence aligned corpora."
  groups: "Aligner"
  nonemboss: "Y"
  executable: "/home/carsten/bin/giza.sh"
]

infile: config [
  comment: "data direct"
  parameter: "Y"
  prompt: "GIZA++ configuration file"
]
infile: source [
  comment: "data direct"
  parameter: "Y"
  prompt: "Source language vocabulary file."
]
infile: target [
  comment: "data direct"
  parameter: "Y"
  prompt: "Target language vocabulary file."
]
infile: sentences [
  comment: "data direct"
  parameter: "Y"
  prompt: "GIZA++ sentences file."
]
infile: dictionary [
  comment: "data direct"
  additional: "Y"
  template: "$$"
  prompt: "A dictionary file in GIZA++ format."
]

outfile: giza [
  additional: "Y"
  parameter: "Y"
  comment: bindata
  extension: "zip"
  default: "giza.zip"
]
```

Bibliography

References

[Abney1991] Abney, S. 1991. Parsing by chunks. *Principle-based parsing*, pages 257–278.

[Abney1997] Abney, S. 1997. Part-of-speech tagging and partial parsing. *Corpus-based methods in language and speech processing*, 2.

[Bahl, Jelinek, and Mercer1990] Bahl, L.R., F. Jelinek, and R.L. Mercer. 1990. A maximum likelihood approach to continuous speech recognition. *Readings in speech recognition*, pages 308–319.

[Baum1972] Baum, L.E. 1972. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, 3(1):1–8.

[Bourigault1992] Bourigault, D. 1992. Surface grammatical analysis for the extraction of terminological noun phrases. In *Proceedings of the 14th conference on Computational linguistics-Volume 3*, pages 977–981. Association for Computational Linguistics.

[Brill1994] Brill, E. 1994. Some advances in transformation-based part of speech tagging. In *Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, pages 722–727. American Association for Artificial Intelligence.

[Brown1996] Brown, M.R. 1996. FastCGI specification. *Open Market Inc., http://fastcgi. idle. com/kit/doc/fcgi-spec. html.*

[Brown et al.1990] Brown, P.F., J. Cocke, S.A.D. Pietra, V.J.D. Pietra, F. Jelinek, J.D. Lafferty, R.L. Mercer, and P.S. Roossin. 1990. A statistical approach to machine translation. *Computational Linguistics*, 16(2):85.

[Brown, Lai, and Mercer1991] Brown, P.F., J.C. Lai, and R.L. Mercer. 1991. Aligning sentences in parallel corpora. In *Proceedings of the 29th annual meeting on Association for Computational Linguistics*, pages 169–176. Association for Computational Linguistics.

[Brown et al.1993] Brown, P.F., V.J.D. Pietra, S.A.D. Pietra, and R.L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

[Chen1993] Chen, S.F. 1993. Aligning sentences in bilingual corpora using lexical information. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 9–16. Association for Computational Linguistics.

[Chiang2005] Chiang, D. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 263–270. Association for Computational Linguistics.

[Chiang et al.2005] Chiang, D., A. Lopez, N. Madnani, C. Monz, P. Resnik, and M. Subotin. 2005. The Hiero machine translation system: Extensions, evaluation, and analysis. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, page 786. Association for Computational Linguistics.

[Christensen et al.2001] Christensen, E., F. Curbera, G. Meredith, and S. Weerawarana. 2001. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[Church1988] Church, K.W. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the second conference on Applied natural language processing*, pages 136–143. Association for Computational Linguistics.

[Cunningham et al.2001] Cunningham, H., D. Maynard, V. Tablan, C. Ursu, and K. Bontcheva. 2001. Developing language processing components with GATE. *GATE v2. 0 User Guide, University of Sheffield*.

[Curbera et al.2002] Curbera, F., M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. 2002. Unraveling the Web services web: an introduction to SOAP, WSDL, andUDDI. *IEEE Internet computing*, 6(2):86–93.

[Davidson and Coward1999] Davidson, J.D. and D. Coward. 1999. Java servlet specification, V2. 2. *Final Release, December*.

[Dempster et al.1977] Dempster, A.P., N.M. Laird, D.B. Rubin, et al. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38.

[Diab, Hacioglu, and Jurafsky2004] Diab, M., K. Hacioglu, and D. Jurafsky. 2004. Automatic tagging of Arabic text: From raw text to base phrase chunks. In *Proceedings of HLT-NAACL 2004: Short Papers on XX*, pages 149–152. Association for Computational Linguistics.

[Ferrucci and Lally2004] Ferrucci, D. and A. Lally. 2004. UIMA: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):348.

[Fielding et al.1999] Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. 1999. Hypertext transfer protocol–HTTP/1.1.

[Franks et al.1999] Franks, J., P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. 1999. RFC2617: HTTP authentication: basic and digest access authentication. *Internet RFCs*.

[Gale and Church1994] Gale, W.A. and K.W. Church. 1994. A program for aligning sentences in bilingual corpora. *Computational linguistics*, 19(1):75–102.

[Gough and Way2003] Gough, N. and A. Way. 2003. Controlled generation in example-based machine translation. In *MT Summit IX*, pages 133–140. Citeseer.

[Gough and Way2004a] Gough, N. and A. Way. 2004a. Example-based controlled translation. In *Proceedings of the Ninth EAMT Workshop*, pages 73–81. Citeseer.

[Gough and Way2004b] Gough, N. and A. Way. 2004b. Robust large-scale EBMT with marker-based segmentation. In *Proceedings of the 10th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-04)*, pages 95–104. Citeseer.

[Gough, Way, and Hearne2002] Gough, N., A. Way, and M. Hearne. 2002. Example-based machine translation via the Web. *Machine Translation: From Research to Real Users*, pages 74–83.

[Green1979] Green, TRG. 1979. The necessity of syntax markers: Two experiments with artificial languages. *Journal of Verbal Learning and Verbal Behavior*, 18(4):481–496.

[Gudgin et al.2001] Gudgin, M., M. Hadley, N. Mendelsohn, J.J. Moreau, H.F. Nielsen, A. Karmarkar, and Y. Lafon. 2001. SOAP Version 1.2. *W3C Working Draft*, 9.

[Gundavaram1996] Gundavaram, S. 1996. *CGI programming on the World Wide Web*. O'Reilly Sebastopol, CA.

[Haas and Brown2004] Haas, H. and A. Brown. 2004. Web Services Glossary. http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211.

[Hearne et al.2007] Hearne, M., J. Tinsley, V. Zhechev, and A. Way. 2007. Capturing translational divergences with a statistical tree-to-tree aligner. In *Proceedings of the 11th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI 07)*, pages 85–94. Citeseer.

[Hunter and Crawford2001] Hunter, J. and W. Crawford. 2001. *Java servlet programming*. O'Reilly Media.

[Ide, Bonhomme, and Romary2000] Ide, N., P. Bonhomme, and L. Romary. 2000. An XML-based Encoding Standard for Linguistic Corpora. In *Proceedings of the Second International Conference on Language Resources and Evaluation*, pages 825–830. Citeseer.

[Kay and Röscheisen1993] Kay, M. and M. Röscheisen. 1993. Text-translation alignment. *Computational Linguistics*, 19(1):121–142.

[Koehn2004] Koehn, P. 2004. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. *Machine translation: From real users to research*, pages 115–124.

[Koehn2005] Koehn, P. 2005. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5. Citeseer.

[Koehn et al.2007] Koehn, P., H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 177–180. Association for Computational Linguistics.

[Ma2006] Ma, X. 2006. Champollion: A robust parallel text sentence aligner. In *Proceedings of the fifth international conference on Language Resources and Evaluation (LREC)*. Citeseer.

[Maamouri et al.2004] Maamouri, M., A. Bies, T. Buckwalter, and W. Mekki. 2004. The penn arabic treebank: Building a large-scale annotated arabic corpus. In *NEMLAR Conference on Arabic Language Resources and Tools*, pages 102–109. Citeseer.

[Marcu and Wong2002] Marcu, D. and W. Wong. 2002. A phrase-based, joint probability model for statistical machine translation. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, page 139. Association for Computational Linguistics.

[Martens2003] Martens, A. 2003. Usability of web services. In *Fourth International Conference on Web Information Systems Engineering Workshops, 2003. Proceedings*, pages 182–190.

[Melamed1996] Melamed, I.D. 1996. A geometric approach to mapping bitext correspondence. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1–12.

[Melamed1997] Melamed, I.D. 1997. A word-to-word model of translational equivalence. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, pages 490–497. Association for Computational Linguistics.

[Molina and Pla2002] Molina, A. and F. Pla. 2002. Shallow parsing using specialized hmms. *The Journal of Machine Learning Research*, 2:595–613.

[Moore2002] Moore, R. 2002. Fast and accurate sentence alignment of bilingual corpora. *Machine Translation: From Research to Real Users*, pages 135–144.

[Nesson, Shieber, and Rush2006] Nesson, R., S.M. Shieber, and A. Rush. 2006. Induction of probabilistic synchronous tree-insertion grammars for machine translation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas (AMTA 2006), Boston, Massachusetts*, pages 8–12. Citeseer.

[Och1995] Och, F.J. 1995. Maximum-Likelihood-Schätzung von Wortkategorien mit Verfahren der kombinatorischen Optimierung. *Studienarbeit, Friedrich-Alexander-Universität, Erlangen-Nürnberg, Germany.*

[Och1999] Och, F.J. 1999. An efficient method for determining bilingual word classes. In *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics*, pages 71–76. Association for Computational Linguistics.

[Och and Ney2000] Och, F.J. and H. Ney. 2000. A comparison of alignment models for statistical machine translation. In *Proceedings of the 18th conference on Computational linguistics-Volume 2*, pages 1086–1090. Association for Computational Linguistics.

[Och and Ney2003] Och, F.J. and H. Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

[Och et al.1999] Och, F.J., C. Tillmann, H. Ney, et al. 1999. Improved alignment models for statistical machine translation. In *Proc. of the Joint SIGDAT Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 20–28.

[Oinn et al.2004] Oinn, T., M. Addis, J. Ferris, D. Marvin, M. Greenwood, T. Carver, M.R. Pocock, A. Wipat, and P. Li. 2004. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics.*

[Oinn et al.2006] Oinn, T., M. Greenwood, M. Addis, M.N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, et al. 2006. Taverna: Lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100.

[Papineni et al.2002] Papineni, K., S. Roukos, T. Ward, and W.J. Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

[Peterson et al.2009] Peterson, D., S. Gao, A. Malhotra, C.M. Sperberg-McQueen, and H.S. Thompson. 2009. W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. http://www.w3.org/TR/2009/WD-xmlschema11-2-20091203.

[Ramshaw and Marcus1995] Ramshaw, L.A. and M.P. Marcus. 1995. Text chunking using transformation-based learning. In *Proceedings of the Third ACL Workshop on Very Large Corpora*, pages 82–94. Cambridge MA, USA.

[Rice et al.2000] Rice, P., I. Longden, A. Bleasby, et al. 2000. EMBOSS: the European molecular biology open software suite. *Trends in genetics*, 16(6):276–277.

[Samuelsson and Volk2006] Samuelsson, Y. and M. Volk. 2006. Phrase alignment in parallel treebanks. In *Proc. of the Fifth Workshop on Treebanks and Linguistic Theories*, pages 91–102.

[Sandvig2004] Sandvig, J.C. 2004. Active Server Pages. *The Internet Encyclopedia*, pages 1–10.

[Senger, Rice, and Oinn2003] Senger, M., P. Rice, and T. Oinn. 2003. Soaplab-a unified Sesame door to analysis tools. In *Proceedings of the UK e-Science All Hands Meeting*, volume 18. Citeseer.

[Simard and Plamondon1998] Simard, M. and P. Plamondon. 1998. Bilingual sentence alignment: Balancing robustness and accuracy. *Machine Translation*, 13(1):59–80.

[Skut and Brants1998] Skut, W. and T. Brants. 1998. Chunk tagger-statistical recognition of noun phrases. *Arxiv preprint cmp-lg/9807007*.

[Stroppa and Way2006] Stroppa, N. and A. Way. 2006. MaTrEx: the DCU machine translation system for IWSLT 2006. In *Proceedings of the International Workshop on Spoken Language Translation*, pages 31–36. Citeseer.

[Varga et al.2005] Varga, D., P. Halácsy, A. Kornai, M.C. Inc, V. Nagy, L. Németh, and V. Trón. 2005. Parallel corpora for medium density languages. *Recent Advances in Natural Language Processing IV: Selected Papers from RANLP 2005*.

[Vogel, Ney, and Tillmann1996] Vogel, S., H. Ney, and C. Tillmann. 1996. HMM-based word alignment in statistical translation. *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 836–841.

[Wahlster2000] Wahlster, W. 2000. *Verbmobil: foundations of speech-to-speech translation*. Springer.

[Way and Gough2003] Way, A. and N. Gough. 2003. wEBMT: developing and validating an example-based machine translation system using the world wide web. *Computational Linguistics*, 29(3):421–457.

[Weaver1955] Weaver, W. 1955. Translation. *Machine translation of languages*, 14:15–23.

[Wu1994] Wu, D. 1994. Aligning a parallel English-Chinese corpus statistically with lexical criteria. In *Annual Meeting – Association for Computational Linguistics*, volume 32, pages 80–87. Association for Computational Linguistics.

[Zhechev and Way2008] Zhechev, V. and A. Way. 2008. Automatic generation of parallel treebanks. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 1105–1112. Association for Computational Linguistics.